Prof. Dr. Erik Jacobson
Fachbereich MND
Fachhochschule Frankfurt am Main

Laborversuche im

R E A L Z E I T L A B O R

zur Lehrveranstaltung "Realzeitsysteme"
im
Studiengang Informatik
der FH Ffm, Fb 2
im
WS 2002/03

Version 1.4 : Mai 2002

Inhalt:

Allgemeine Hinweise zum Laborbetrieb

Versuchskurzbeschreibungen

Übung 1a: Messung der Laufzeiten von Anweisungen in einer höheren Programmiersprache
Übung 1b: Messung der Laufzeiten bei Array-Bearbeitung in einer höheren Programmiersprache
Übung 1c: Messung der Laufzeiten von Instruktionen in einer maschinennahen Programmiersprache
Übung 2a: Beobachtungen an der Terminalschnittstelle mit einem Oszilloskop
Übung 2b: Programmierung des Eingabeechos an einer Terminalschnittstelle (COM-1)
Übung 2c: Kommunikation zwischen 2 Rechnern über die Terminalschnittstelle (COM-1)
Übung 3a: Programmierungen mit Timer-Funktionen
Übung 3b: Programmierung zum Task-Scheduling
Übung 3c: Programmierung von Interprozeßkommunikation
Übung 4a: Prozeßsteuerung zum Kochen eines Eies über eine Digitalschnittstelle
Übung 4b: Beobachtungen an der Busschnittstelle des Prozessors 68 000 mit einem Logikanalysator
Übung 4c: Modellierung einer Regelstrecke mit RESI
Übung 5a: Programmierung, Test und Eichung eines digitalen Photometers
Übung 5b: Programmierung, Test und Eichung eines digitalen Thermometers
Übung 5c: Programmierung und Test für die Erfassung von Schallereignissen
Übung 5d: Programmierung und Test für die Steuerung eines Plotters

A. Allgemeines zur Versuchsdurchführung:

- Die Aufgaben werden in Gruppen zu je 2 Personen bearbeitet.
- Die Bearbeitung erfolgt auf den Rechnern im Realzeitlabor BCN 103 unter LINUX, MS-DOS oder Lynx/OS
- Jede Gruppe bearbeitet mindestens je eine Aufgabe aus den Aufgabenblöcken (1a ..c), (2a, ..b), (3a, ..c), (4a ..c), (5a .. d)
- Die Versuchsbeschreibungen und die ausgeteilten Laborblätter dienen der Vorbereitung, der Durchführung und der Nachbereitung.
-- Zur Vorbereitung sind insbesondere die entsprechende Literatur zu lesen, einige Diagramme zu skizzieren und Programme zu erstellen.
-- Für die Programme sind Kurzbeschreibungen (Formblatt nach DIN 66 230), Struktogramme (nach Nassi-Shneiderman) und Quellcode in verschiedenen Programmiersprachen zu erstellen.
-- Alle größeren Programme sind modular aufzubauen und arbeitsteilig zu erstellen. Alle wichtigen Module sollten so gestaltet sein, daß sie einzeln getestet werden können, um Fehler schneller eingrenzen zu können oder Unklarheiten zu beseitigen. Daher ist eine gute funktionelle Gliederung unerläßlich.
- Bei allen Versuchen ist darauf zu achten, daß die Versuchsanordnung zu Beginn in einen definierten Zustand gebracht, d.h. initialisiert wird.
- Während der Versuchsdurchführung sind anhand der Versuchsbeschreibungen die Beobachtungen durchzuführen und Messungen von Hand aufzuzeichnen oder auszudrucken.
- Bei der Nachbereitung werden Werte, die während des Versuchs nicht mehr berechnet werden konnten, ausgerechnet und Deutungen von Versuchsergebnissen anhand der Literatur ausgearbeitet.

Als Ausarbeitungen abzugeben sind:

1. Benutzerhandbuch (Arbeitsanleitung)
   Bedienungsanleitung für die Programme, einschließlich manueller Weiterverarbeitungen
   Es muß alles enthalten, was zur Erstellung der Ergebnisse notwendig ist.
   - Der Benutzer kann und darf nicht compilieren oder assemblieren - !!!!

Übersicht über die Übungsaufgaben

Übung 1a: Messung der Laufzeiten von Anweisungen in einer höheren Programmiersprache
Mit einfachsten Mitteln (Armbanduhr) sind die Laufzeiten von Anweisungen in einer höheren Programmiersprache, z.B. "C", PASCAL, FORTRAN oder BASIC möglichst genau zu bestimmen.

Übung 1b: Messung der Laufzeiten bei Array-Bearbeitung in einer höheren Programmiersprache
Mit einfachsten Mitteln (Armbanduhr) sind die Laufzeiten für die Bearbeitung eines 2-dimensionalen Array (n*n) in einer höheren Programmiersprache, z.B. "C", PASCAL, FORTRAN oder BASIC, zu bestimmen und die Unterschiede bei spaltenweiser und bei zeilenweiser Abarbeitung zu vergleichen.

Übung 1c: Messung der Laufzeiten von Instruktionen in einer maschinennahen Programmiersprache
Mit einfachsten Mitteln (Armbanduhr) sind die Laufzeiten von Anweisungen in einer maschinennahen Programmiersprache (Assembler) möglichst genau zu bestimmen.

Übung 2a: Beobachtungen an der Terminalschnittstelle mit einem Oszilloskop
- Datendarstellung an der RS 232C Schnittstelle (V.24, V.28, V.4)
- Datenraten
- Fehlanpassungen

Übung 2b: Programmierung des Eingabeechos an einer Terminalschnittstelle
Für ein Terminal (VT 100), das an einen serielle Schnittstelle angeschlossen wird, soll das von der Tastatur eingegebene Zeichen über eine einfache Prozedur auf dem Bildschirm dargestellt werden. Es sollen keine Systemfunktionen verwendet werden, sondern Zugriffe auf die Register des entsprechenden Logikbausteins (UART 8250)

Übung 2c: Kommunikation zwischen 2 Rechnern über die Terminalschnittstelle COM-1
Programmierung der Datenübertragung im Gegensprechverkehr (Full Duplex, FDX). Zeichen, die am Terminal eines Rechners eingegeben werden, sollen am Bildschirm des anderen Rechners erscheinen.

Übung 3a: Programmierungen mit Timer-Funktionen
Messung der Reaktionszeit für eine Tasteneingabe nach zufallsgesteuerter Aufforderung.

Übung 3b: Programmierung zum Task-Scheduling
Unter Programmkontrolle sollen mehrere Tasks gestartet und durch geeignete Tastatureingaben beendet werden. Die Taskaktivitäten sind auf dem Konsolterminal darzustellen.

Übung 3c: Programmierung von Interprozeßkommunikation
Unter Programmkontrolle sollen mehrere Tasks gestartet und durch geeignete Tastatureingaben beendet werden. Zwischen den Tasks soll ein Datenaustausch stattfinden. Die Task-Aktivitäten sind auf dem Konsolterminal darzustellen.

Übung 4a: Prozeßsteuerung zum Kochen eines Eies über die Terminalschnittstelle
Mit der Digitalen Ein/Ausgabeschnittstelle eine Prozeßrechners oder mit der RS 232C Schnittstelle soll eine Meßwerterfassung (Schließen eines Bimetallschalters) und eine Steuerwertausgabe (Ein/Ausschalten eines Kochers) durchgeführt werden, um ein 6-Minuten Ei zu kochen.

Übung 4b: Modellierung einer Regelstrecke mit RESI
Es ist eine Regelstrecke für ein vorgegebenes System zu modellieren und in der Simulation zu testen.

Übung 4c: Beobachtungen an der Busschnittstelle des Prozessors 68 000 mit einem Logikanalysator
Es sind die Bussignale des Prozessors MC 68 000 für unbekannte und bekannte Programmabläufe zu beobachten und zu interpretieren. Hieraus sind Laufzeiten von Maschineninstruktionen zu bestimmen.

Übung 5a: Programmierung, Test und Eichung eines digitalen Photometers
Mit Hilfe der Analogeingabe eines Prozeßrechners sind die Signale einer Solarzelle zu erfassen und als digitale Helligkeitswerte on-line auf dem Bildschirm und off-line als Meßkurve darzustellen.

Übung 5b: Programmierung, Test und Eichung eines digitalen Thermometers
Mit Hilfe der Analogeingabe eines Prozeßrechners sind die Signale einer Thermowiderstands (NTC) zu erfassen und als digitale Temperaturwerte on-line auf dem Bildschirm und off-line als Meßkurve dar zustellen.

Übung 5c: Programmierung und Test für die Erfassung von Schallereignissen
Mit Hilfe der Analogeingabe eines Prozeßrechners sind die Signale eines Mikrofons zu erfassen und als Zeitverlauf und Frequenzspektrum on-line auf dem Bildschirm und off-line als Meßkurve darzustellen.

Übung 5d: Programmierung und Test für die Steuerung eines Plotters
Mit Hilfe der Analogausgabe eines Prozeßrechners ist ein XY-Schreiber so zu steuern, daß vorgegebene Kurven und Texte dort dargestellt werden.

Aufgabe 1a: Messung der Laufzeiten von Anweisungen in einer höheren Programmiersprache

Ziel der Aufgabe:
Bestimmen Sie mit einfachsten Mitteln (Armbanduhr) die Laufzeiten von Anweisungen in einer höheren Programmiersprache, z.B. "C", PASCAL, FORTRAN oder BASIC:

- Zuweisungsanweisung mit Speicherzugriffen
- Rechenoperationen wie Addition, Subtraktion, Multiplikation, Division
- Unterprogrammsprung
- Zugriffe auf Datenträger (Festplatte oder Diskette)

Suchen Sie sich selber geeignete Anweisungen aus der von Ihnen gewählten Programmiersprache ("C", PASCAL, FORTRAN oder BASIC) aus.

Aufgabenstellung:
Erstellen Sie Ihr Programm samt Struktogramm und Dokumentation.
Tragen Sie in das Arbeitsblatt die untersuchten Anweisungen (a, b, c, d) ein.

Hinweise:
Erstellen Sie vor Übungsbeginn ein Programm samt Struktogramm.
Verwenden Sie in Ihrem Programm eine genügend oft durchlaufene Schleife mit der und ohne die zu untersuchende Anweisung.
Schätzen Sie ab, welche Laufweite Ihre Schleifen haben müssen, um auch mit Hilfe einer sekundengenauen Zeitmessung (Armband- oder Systemuhr) eine genügend große Genauigkeit zu erhalten.

Zusatzaufgabe:
Führen Sie dieselbe Übung auf einem anderen Rechner oder in einer anderen Programmiersprache durch.

D.1 Arbeitsblatt zur Aufgabe 1a: Verarbeitungszeiten

Namen: ..... Datum: .....

.....

.....

Rechnerkonfigurationen

1. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

2. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

3. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

Programmiersprache: 1.: ..... 2.: .....

Anweisung:

a)

b)

c)

d)

Meßwerte:

	Rechner 1 Sprache 1	Rechner 2 Sprache 2
Leerschleife (in msec) (mit ..... Durchläufen)		
Anweisungszeiten (in ns/Instr.)		
a)	.....	.....
b)	.....	.....
c)	.....	.....
d)	.....	.....

Aufgabe 1b: Messung der Laufzeiten bei Array-Bearbeitung in einer höheren Programmiersprache

Ziel der Aufgabe:

Bestimmen Sie mit einfachsten Mitteln (Armbanduhr) die Laufzeiten für die Bearbeitung eines 2-dimensionalen Array (n*n) in einer höheren Programmiersprache, z.B. "C", PASCAL, FORTRAN oder BASIC, und vergleichen Sie die Unterschiede bei spaltenweiser und bei zeilenweiser Abarbeitung.

Aufgabenstellung:

Erstellen Sie Ihr Programm samt Struktogramm und Dokumentation.
Tragen Sie in das Arbeitsblatt die Bearbeitungszeiten als Funktion der Arraygröße n auf.

Hinweise:

- verwenden Sie eine genügend oft durchlaufene Schleife mit der spaltenweisen und der zeilenweisen Bearbeitung und bilden Sie die Differenz.
- Füllen Sie das Array als Vorbereitung mit einer aufsteigenden Folge von Zahlen und bilden als Bearbeitung hierüber die Summe.
- Steigern Sie die Größe des Array sukzessive und beobachten Sie auch etwaige Sprünge !

Zusatzaufgabe:

Führen Sie dieselbe Übung auf einem anderen Rechner oder in einer anderen Programmiersprache durch.

D.2 Arbeitsblatt zur Übung 1b: Bearbeitungszeiten

Namen: ..... Datum: .....

.....

.....

Rechnerkonfigurationen

1. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

2. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

3. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

Programmiersprache: 1.: ..... 2.: .....

Laufzeit (in ms)

Aufgabe 1c: Messung der Laufzeiten von Instruktionen in einer maschinennahen Programmiersprache

Ziel der Aufgabe:
Bestimmen Sie mit einfachsten Mitteln (Armbanduhr) die Laufzeiten von Anweisungen in einer Assemblersprache:

- Löschen: CLEAR A
- Datentransfer: MOVE A, B
- Vergleich: COMPARE A, B

Suchen Sie sich selber geeignete Instruktionen aus dem Instruktionssatz des verwendeten Rechners aus.
Verwenden Sie für mindestens eine Instruktion (a) unterschiedliche Adressierungsarten ($m \geq 2$)

Aufgabenstellung:
Erstellen Sie Ihr Programm samt Struktogramm und Dokumentation.
Tragen Sie in das Arbeitsblatt die untersuchten Instruktionen (a_m, b, c) und die verwendeten Rechnerkonfigurationen n ($n \geq 2$) ein.

Hinweise:
Erstellen Sie vor Übungsbeginn ein Programm samt Struktogramm.
Verwenden Sie in Ihrem Programm eine genügend oft durchlaufene Schleife mit der und ohne die zu untersuchende Anweisung.
Schätzen Sie ab, welche Laufweite Ihre Schleifen haben müssen, um auch mit Hilfe einer sekundengenauen Zeitmessung (Armband- oder Systemuhr) eine genügend große Genauigkeit zu erhalten.

D.3 Arbeitsblatt zur Aufgabe 1c: Instruktionszeiten

Namen: ..... Datum: .....

.....

.....

Rechnerkonfigurationen

1. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

2. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

3. Rechner: ..... Prozessor: ..... Taktfrequenz: .....

Instruktionen:

a1)

a2)

a3)

a4)

b)

c)

d)

Meßwerte:

	Rechner 1	Rechner 2	Rechner 3
Leerschleife (in sec) (mit ..... Durchläufen)			
Instruktionszeiten (in ns/Instr.)			
a1)	.....	.....	.....
a2)	.....	.....	.....
a3)	.....	.....	.....
a4)	.....	.....	.....
b)	.....	.....	.....
c)	.....	.....	.....
d)	.....	.....	.....

Übung 2a: Beobachtungen an der Terminalschnittstelle mit einem Oszilloskop

Ziel dieser Aufgabe:
Aus den beobachteten Signalen sollen die übertragenen Zeichen und die eingestellten Parameter der RS-232C-Schnittstelle bestimmt werden. Themen:

- Datendarstellung an der RS 232C Schnittstelle (V.24, V.28, V.4)
- Datenraten
- (Fehl)anpassungen

0. Vorbereitung:

Programm entwerfen, ASCII-Tabelle besorgen und den voraussichtlichen Signalverlauf aufzeichnen!

1. Inbetriebnahme der Versuchsanordnung:

1.1 An der **Ausgangsbuchse (COM 1 - Port)** den Teststecker anschließen und **vor Einschalten** des Rechners überprüfen lassen.
1.2 Das **Oszilloskop** (Grundig MO30, Philips PM 3208 oder PM 3382A) am Netzschalter einschalten.
Die Schalterstellungen, die beachtet werden müssen, sind auf den Beiblättern (E1 bis E3) angegeben.
- Der Tastkopf des Oszilloskops ist an den Teststecker (V.24-Tester) zuerst folgendermaßen anzuschließen:
  - Masse an Leitung 7 (GND) = Signalerde (ground)
  - Kanal CH1 an Leitung 3 (TxD) = Sendedaten (transmitted data)
- Zur Kontrolle der Sendedaten kann eine Rückführung über Leitung 2 (RxD) angeschlossen und per Software geprüft werden. Falls ein VT-100 Terminal frei ist, kann dieses zur äußeren Kontrolle angeschlossen werden.

1.3 Das **Terminal** am Netzschalter einschalten.
Bei geeigneter Einstellung des Terminalbildschirms (SETUP) erfolgt ein automatischer Zeilenumbruch. Sonst wird am Ende der Zeile das letzte Zeichen immer wieder überschrieben.
Anmerkung: Diese Aufgabe kann auch ohne Terminal durchgeführt werden.

2. Erstellung und Eingabe eines Testprogramms

Erstellen Sie ein Testprogramm, das folgende Bedingungen erfüllt:

- Der Zugriff auf die Terminalschnittstelle erfolgt über eine Systemfunktion wie "write".
- Das Programm läuft in einer Endlosschleife.
- Von der Konsoltastatur soll das Zeichen eingegeben werden, das auf der Leitung erscheinen soll.
- Der Zeitabstand der ausgegebenen Zeichen muß variabel sein (entweder über eine Timer-Funktion oder über eine Warteschleife)
- Der Zeitabstand (= Wartezeit) soll möglichst on-line verändert werden können; andernfalls ist u.U. immer einen Neucompilation nötig, die relativ viel Zeit kostet.
- Es muß eine ESCAPE-Funktion zum Beenden des Programms vorhanden sein (CTRL/C, o.ä.).

3. Darstellung und Aufzeichnung der Signale

Stellen Sie die Zeitbasis des Oszilloskops zunächst so ein, daß eine Programmschleife möglichst den gesamten Darstellungsbereich ausfüllt, d.h. so daß 2 Zeichenausgaben zu sehen sind. Halten Sie dabei einen geeichten Zeitmaßstab (am Oszilloskop) ein.

- Betrachten Sie das Ausgabesignal genauer durch Wahl eines anderen Zeitmaßstabs und/oder einer kürzeren Wartezeit (swo).
  Notieren Sie sich den Wert des verwendeten Schleifenzählers oder des Timers.
- Richten Sie Ihren Zeitmaßstab (variabel) so ein, daß ein Signalelement genau eine Rastereinheit ausfüllt.
- Zeichnen Sie die Signale maßstäblich (mit Zeitskala und Pegeln) für folgende Zeichen auf:
  A, a, 1, 2, 3, 4
- Bestimmen Sie die Anzahl der Datenbits und die Einstellungen für das Prüfbit.
- Bestimmen Sie die Zeitdauer eines Signalelements in einem geeichten Zeitmaßstab.
- Bestimmen Sie die Anzahl der Stopbits, indem Sie die Wartezeit auf 0 verkürzen.
- Erhöhen Sie die Wartezeit solange, bis sie sich im Zeichenabstand wieder bemerkbar macht; vergleichen Sie die Wartezeit mit der nominellen Ausgaberate (Baud-Rate)
- Messen Sie die zeitliche Dauer des Ausgabesignals und berechnen Sie daraus die maximale Ausgaberate (in Zeichen pro Sekunde, cps).

4. Beobachtung von Fehlanpassungen (Zusatzaufgabe)

- Schließen Sie ein VT-100 Terminal an die COM-Schnittstelle an
- Stellen Sie mit dem SETUP die richtigen Kommunikationsparameter (Baudrate, Datenbits, Parität, Stopbits) ein und beobachten Sie den korrekten Empfang der vom Rechner gesendeten Zeichen.
- Ändern Sie im SETUP des Terminals die Empfangs-Baudrate, zuerst auf den doppelten, dann auf den halben Wert
- Beobachten Sie die Änderung der Darstellung auf dem Terminal; in der Regel werden sogenannte Schmierzeichen (ausgefüllte Rechtecke oder umgedrehte Fragezeichen) erscheinen, die einen Fehler in der Datenübertragung signalisieren.
- Bei Wahl geeigneter Eingabewerte von der Tastatur können lesbare Zeichen entstehen. Zeichnen Sie die entsprechenden Signalfolgen maßstabsgetreu auf und interpretieren Sie die Darstellung.
- Stellen Sie zum Schluß im SETUP die Standard-Baudrate wieder ein (9600 Baud).

5. Ausarbeitung

- Programmdokumentation (Quellcode, Struktogramme, etc)
- Arbeitsanleitung (Benutzerhandbuch)
- Beschreibung der Beobachtungen bei der Durchführung

D.4 Arbeitsblatt zur Aufgabe 2a: Start-Stop-Prozedur

Namen: ..... Datum: .....

.....

..... Rechner: .....

Zeitmaßstab: ..... msec/div

- Signale (maßstäblich mit Zeitskala) für folgende Zeichen:

A

a

1

2

3

4

- Anzahl der Datenbits.....
- Einstellung des Prüfbits..
- Anzahl der Stopbits.....

- Signalpegel "0" = ... V; "1" = ... V
- Anstiegszeit (Flankensteilheit): .....

- Zeitdauer eines Signalelements (bits): ..... msec
- Zeitliche Dauer des Ausgabesignals ..: ..... msec
- Maximale Ausgaberate (in Zeichen pro Sekunde): ..... cps

- Der optimale Wert für die Wartezeit: .....

- Effekte, die bei zu kurzer Pause zwischen 2 Zeichen störend auf dem Bildschirm erscheinen:

.....

- Fehlanpassungen mit Sender-Baudrate: .....

Empfänger-Baudrate: .....

Sendezeichen: A, a

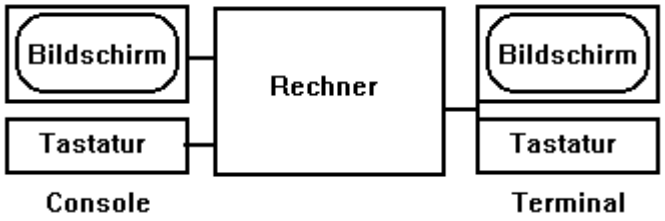
Empfangszeichen auf dem Bildschirm: ....

Übung 2b: Programmierung des Eingabeechos an einer Terminalschnittstelle

Programmierung des Terminal-Echo mit Polling und mit Interrupt

Ziel dieser Aufgabe:

Diese Aufgabe soll verdeutlichen, daß eine Terminaleingabe von der Tastatur nicht automatisch auch auf dem Bildschirm erscheint, sondern erst durch geeignete Software.



0. Vorbereitung:

- Strukturen durch Petri-Netze definieren.
- Geeignete Testprogramme (Struktogramme, Code) erstellen.

1. Grundlagen:

Eine Eingabe von der Tastatur eines Terminal wird nicht automatisch auf dem Bildschirm dargestellt, sondern es muß durch eine geeignete Prozedur eingelesen und auf dem Bildschirm wieder ausgegeben werden. Dieses Verfahren nennt man Echo.

Zu seiner Realisierung werden 2 unterschiedliche Methoden benutzt:

- **Polling-Verfahren**, d.h. die ständige Abfrage des Tastaturpuffers, bzw. des zugehörigen Control-Registers (R-CSR) durch das aktive Programm, und Kopieren des Eingabewerts in das Ausgaberegister
- **Interrupt-Betrieb**, bei dem ein laufendes Programm unterbrochen, sobald ein Ereignis eintritt, z.B. ein Tastendruck, und eine geeignete Routine aktiviert wird, die das eingegebene Zeichen als Echo auf dem Bildschirm ausgibt.
- Beide Arten können auch kombiniert werden.
- Im neuen Realzeitlabor wird derzeit nur das Polling-Verfahren angewandt, da für den Interrupt-Betrieb "root"-Befugnisse notwendig wären.
- Das Polling-Verfahren kann auch auf Hochsprachenebene eingesetzt werden, ohne die Schnittstellenregister zu verwenden.

2. Inbetriebnahme der Versuchsanordnung:

Für diese Aufgabe ist an den COM-Port des LINUX-Rechners ein VT-100 Terminal anzuschließen; am Terminal sind die notwendigen Schnittstellenparameter einzustellen

3. Erstellung und Eingabe des Testprogramms

Erstellen Sie ein Testprogramm, das folgende Bedingungen erfüllt:

- Das Programm läuft in einer Endlosschleife.
- Der Zugriff auf die Terminalschnittstelle erfolgt über die Register des UART (s. Anlage)
  oder über Systemfunktionen wie "read" und "write", "getc" oder "putc"
- Das auf der Terminal-Tastatur eingegebene Zeichen erscheint unter Programmkontrolle auf dem Terminal-Bildschirm (eine Veränderung des Zeichens soll im Programm eingeplant werden !).
- In der Endlosschleife ist eine Warteschleife einzubauen, die andere Rechneraufgaben simuliert. Die Wartezeit soll variabel sein (entweder über eine Timer-Funktion oder über einen Schleifenzähler)
- Die Wartezeit soll über die Consol-Tastatur verändert werden können. Eine Neucompilation darf nicht notwendig sein, da sie relativ viel Zeit kostet.
- Es muß eine ESCAPE-Funktion zum Beenden des Programms vorhanden sein (CTRL/C, o.ä.).

4. Durchführung der Messungen

- Stellen Sie mit dem SETUP die richtigen Kommunikationsparameter (Baudrate, Datenbits, Parität, Stopbits) ein und beobachten Sie den korrekten Empfang der vom Rechner gesendeten Zeichen.
- Die Reaktionen des Terminals auf Terminaleingaben sind für verschiedene Warteschleifen zu beobachten und zu beschreiben.

5. Ausarbeitung

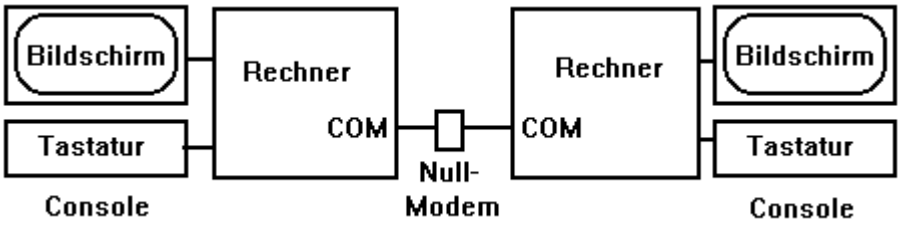
- Programmdokumentation (Quellcode, Struktogramme, etc)
- Arbeitsanleitung (Benutzerhandbuch)
- Beschreibung der Beobachtungen bei der Durchführung

Übung 2c: Kommunikation zwischen 2 Rechnern über die Terminalschnittstelle COM-1

Programmierung der Datenübertragung im Gegensprechverkehr (Full Duplex, FDX)

Ziel dieser Aufgabe:

Diese Aufgabe soll verdeutlichen, wie ein Rechner auch als Terminal benutzt werden kann. Und wie über diese Schnittstelle auch Daten übertragen werden können.
Die Zeichen, die am Terminal eines Rechners eingegeben werden, sollen am Bildschirm des anderen Rechners erscheinen.



0. Vorbereitung:

- Strukturen durch Petri-Netze definieren.
- Geeignete Testprogramme (Struktogramme, Code) erstellen.
- Geeignetes Null-Modem-Kabel konfigurieren.

1. Grundlagen

Tastatureingaben werden normalerweise auf dem Bildschirm des Terminals geechot (vgl. Aufgabe 2b). In dieser Aufgabe sollen die Zechen, die an einem Rechner eingegeben werden, auf dem Bildschirm eines anderen Rechners (Partner) erscheinen.
Dazu sind auf beiden Rechnern identische Programm zu installieren, die eine Kommunikation im Gegensprechverkehr (Full Duplex, FDX) ermöglichen.
Die Zeichen, die an der Tastatur eingegeben werden, werden auf die COM-Schnittstelle geschickt und am anderen Rechner an der COM-Schnittstelle empfangen und auf dem Bildschirm ausgegeben.
Hierfür werden Systemfunktionen wie "getc" und "putc" verwendet.

2. Inbetriebnahme der Versuchsanordnung:

Für diese Aufgabe müssen 2 Rechner über ihre COM-Schnittstellen miteinander verbunden werden.
Hierfür sind geeignete Kabel und ein "Nullmodem" zu verwenden.
Das "Nullmodem" hat im wesentlichen die Aufgabe, ein Paar von Modem zu ersetzen. Dazu sind Sende- und Empfangsleitungen kreuzweise miteinander zu verbinden. Ebenso die Steuerleitungen (RTS, CTS, DSR, DTR)

3. Erstellung und Eingabe des Testprogramms

Auf beiden Rechner ist jeweils das gleiche Programm zu erstellen, zu testen und in Betrieb zu nehmen.
Es ist in einer geeigneten (maschinennahen) Programmiersprache zu schreiben; vorzugsweise in "C".

4. Durchführung

Die Reaktionen der Rechner auf Consol-Eingaben Gegensprechverkehr (Full Duplex, FDX) sind zu beobachten und zu beschreiben.

5. Ausarbeitung

- Programmdokumentation (Quellcode, Struktogramme, etc)
- Arbeitsanleitung (Benutzerhandbuch)
- Beschreibung der Beobachtungen bei der Durchführung

6. Zusatzaufgabe

Beobachten Sie den Datenverkehr am Nullmodem mit einem Oszilloskop (vgl. Aufgabe 2a)

Übung 3a: Programmierungen mit Timer-Funktionen

Messung der Reaktionszeit für eine Tasteneingabe nach zufallsgesteuerter Aufforderung.

Ziel dieser Aufgabe:
Programmieren von Zeitvorgaben(ausgaben) und Zeitmessung(eingaben).

0. Aufgabenstellung

Um die Reaktionszeit einer Testperson zu messen, werden in unregelmäßigen Abständen (einstellbare Intervalle der Wartezeiten von ca. 1 bis 60 Sekunden) Eingabeaufforderungen erzeugt, hierbei ist dafür zu sorgen, daß vorzeitige Eingaben abgewiesen werden. Für die darauf folgende Eingabe ist die benötigte Reaktionszeit zu messen und für eine nachfolgende Auswertung zu speichern.
In der Auswertung sind zu bestimmen:
- Für die Wartezeit (w) Mittelwert und Varianz (Minimal- und Maximalwert werden vorgegeben)
- Für die Reaktionszeit (r) Mittelwert und Varianz, Minimal- und Maximalwert

1. Vorbereitung

- Strukturen durch Petri-Netze definieren.
- Geeignete Testprogramme (Struktogramme, Code) erstellen.

2. Inbetriebnahme der Versuchsanordnung

Zur Durchführung dieses Versuchs ist jeder beliebige PC verwendbar, auf dem geeignete Timer-Funktionen verfügbar sind.

3. Erstellung des Testprogramms

Das Programm ist auf einem der Unix (LINUX oder Lynx/OS) - Rechner im Realzeitlabor zu erstellen und zu installieren.

4. Durchführung

Machen Sie mehrere Versuche, mit unterschiedlichen Parametern um die optimalen Werte herauszufinden für:
- die Wartezeiten: Mittelwert und Varianz bzw. Minimal- und Maximalwerte
- die Anzahl der Meßzyklen (10 bis 100)

5. Ausarbeitung

- Programmdokumentation (Quellcode, Struktogramme, etc)
- Arbeitsanleitung (Benutzerhandbuch)
- Meßergebnisse

6. Zusatzaufgabe

Bestimmen Sie eine etwaige Korrelation zwischen Wartezeit (w) und Reaktionszeit (r) numerisch und in Form eines Diagramms r(w).

Übung 3b: Programmierung zum Task-Scheduling

Ziel dieser Aufgabe:

Unter Programmkontrolle sollen mehrere Tasks gestartet und durch geeignete Tastatureingaben beendet werden. Die Task-Aktivitäten sind auf dem Konsolterminal darzustellen.

0. Aufgabenstellung

Es soll ein Programm erstellt werden, das beim Ablauf durch Fork-Prozesse 5 neue Tasks (A...E) erzeugt. Diese Tasks sollen im Wechsel ablaufen. Jeder Taskwechsel ist auf der Console mit Zeitstempel kenntlich zu machen. Außerdem ist vorzusehen, daß jede einzelne Task (A...E) während ihrer Laufzeit durch eine Consol-Eingabe beendet werden kann, ohne daß der Gesamtprozeß (Job) abgebrochen wird.

1. Vorbereitung:

- Strukturen durch Petri-Netze definieren.
- Geeignete Testprogramme (Struktogramme, Code) erstellen.

2. Inbetriebnahme der Versuchsanordnung:

Zur Durchführung dieses Versuchs ist jeder Rechner verwendbar, der Multi-Tasking erlaubt.

3. Erstellung des Testprogramms

Das Programm ist auf einem der LINUX - Rechner im Realzeitlabor zu erstellen und zu installieren.
Beginnen Sie mit 2 Kindprozessen (A, B) und erweitern Sie dann die Menge der Kindprozesse auf 5.

4. Durchführung

Der Ablauf des Programms ist zu dokumentieren. Am einfachsten durch Umleiten der Standardausgabe in eine Datei.

5. Ausarbeitung

Programmdokumentation, Bedienungsanleitung, Ablaufprotokoll

6. Zusatzaufgabe

Am Anfang des Programms soll durch Benutzereingabe ausgewählt werden können, ob der Taskwechsel zyklisch erfolgt oder rein zufällig.

Übung 3c: Programmierung von Interprozeßkommunikation

Ziel dieser Aufgabe:

Unter Programmkontrolle sollen mehrere Tasks gestartet und nach einer vorgegebenen Anzahl von Durchläufen beendet werden. Zwischen den Tasks soll ein Datenaustausch stattfinden. Die Taskaktivitäten sind auf dem Konsolterminal darzustellen.

0. Aufgabenstellung

Es soll ein Programm erstellt werden, das beim Ablauf durch Fork-Prozesse 4 neue Tasks (A...D) erzeugt, von denen jeweils 2 Erzeuger und 2 Verbraucher darstellen. Die Erzeuger-Tasks sollen im Wechsel ablaufen und Nachrichten über eine Pipe an die Verbraucher-Tasks senden. Das Absenden durch die Erzeuger und die Annahme durch die Verbraucher sind auf der Console mit Zeitstempel darzustellen.

1. Vorbereitung:

- Strukturen durch Petri-Netze definieren.
- Geeignete Testprogramme (Struktogramme, Code) erstellen.

2. Inbetriebnahme der Versuchsanordnung:

Zur Durchführung dieses Versuchs ist jeder Rechner verwendbar, der Multi-Tasking erlaubt.

3. Erstellung des Testprogramms

Das Programm ist auf einem der LINUX - Rechner im Realzeitlabor zu erstellen und zu installieren.
Beginnen Sie mit jeweils einem Erzeuger- und einem Verbraucherprozeß.

4. Durchführung

Der Ablauf des Programms ist zu dokumentieren. Am einfachsten durch Umleiten der Standardausgabe in eine Datei.

5. Ausarbeitung

Programmdokumentation, Bedienungsanleitung, Ablaufprotokoll

6. Zusatzaufgabe

Führen Sie die Übung so durch, daß die 4 Tasks als 4 eigenständige Programme (aus 4 Fenstern) gestartet werden.

Übung 4a: Prozeßsteuerung zum Kochen eines Eies

Ziel dieser Aufgabe:
Erstellung einer einfachen digitalen Regelung

0. Aufgabenstellung

Mit Hilfe eines Computers soll ein 6-Minuten Ei gekocht werden.
Als Prozeßglieder sind vorhanden:

1. ein Bimetallschalter, der bei ca. 90°C öffnet; er besitzt keine eigene Spannungsversorgung, sie muß ihm bereitgestellt werden !!!
2. ein Relais für die Heizung eines Kochtopfs (Tauchsieder)

Es gilt eine Heizung (Kochplatte oder Tauchsieder) mit Hilfe eines Bimetallschalters so zu regeln, daß keine zu starke Erhitzung und kein Überkochen des Wassers eintritt.
Durch geeignete Meldungen soll der Anwender zum Bedienen des Prozesses aufgefordert werden: z.B. zur Inbetriebnahme des Prozesses, zum Einlegen und Herausnehmen des Eies.
Im Vorlauf des Steuerprogramms oder als eigenständiges Programm ist ein Test der Ein/Ausgabeschnittstelle durchzuführen (s. 3.1) !!!!!!!!!!!!

Zur Realisierung stehen 2 unterschiedliche Systeme zur Verfügung:
A: Linux-Rechner mit einer seriellen COM-Schnittstelle. Als Behelfsschnittstelle werden hier die Steuersignale der RS-232 Schnittstelle verwendet, die über die Register des UART erzeugt und empfangen werden (vgl. Aufgabe 2b). Hiermit wird ein digitales Ausgangssignal (z.B. mit RTS oder DTR) für ein Relais erzeugt, das die Heizung schaltet, und ein digitales Signal vom Bimetallschalter (mit CTS oder DSR) empfangen.
*Der Zugriff erfolgt über Systemfunktionen oder über die UART-Register im IO-Bereich, für die "root"-Privilegien erforderlich wären. Zum Zugriff nichtprivilegierter Benutzer steht unter **LINUX** eine kleine Bibliothek **USER_COM** zur Verfügung, die mit eingebunden werden muß:*

```
/*Test fuer user_COM, needs "insmod userio_module" by root */
extern unsigned char user_COM1_in(unsigned short port);
extern unsigned char user_COM2_in(unsigned short port);
extern int user_COM1_out(unsigned short port, unsigned char data);
extern int user_COM2_out(unsigned short port, unsigned char data);
```

B: LYNX-Systeme mit einer Digitalen Ein-Ausgabe-Schnittstelle (DPIO 32), die über den VME-Bus angeschlossen ist. Näheres zu dieser Schnittstelle im entsprechenden Manual (s. Hardcopy).

1. Vorbereitung:

- Ablaufstrukturen durch Zustandsdiagramme definieren, dabei sind auch Störeinflüsse und -ereignisse zu berücksichtigen.
- Geeignete Testprogramme (Struktogramme, Code) erstellen.

2. Erstellung des Programms

Für den Ablauf ist etwa folgendes Rahmenprogramm vorzusehen:

- Vor dem Start des eigentlichen Steuerungsprogramms soll ein Test der Hardware-Schnittstelle interaktiv mit dem Benutzer durchgeführt werden
- Aus Sicherheitsgründen (welchen ?) muß vor dem Anschluß der Heizung ans Netz die Funktionsfähigkeit der angeschlossenen Prozeßperipherie (Relais und Bimetallschalter) überprüft werden.
- Zum Start des Kochvorgangs soll eine Bestätigung vom Benutzer angefordert werden, damit nicht ein Start versehentlich erfolgen kann.
- Nach dem Erreichen der Solltemperatur wird der Anwender zum Einlegen eines Eies aufgefordert, was dieser dann auch bestätigen muß. Die benötigte Aufheizzeit ist auszugeben.
- Das Wasser soll dann unter kontrollierten Bedingungen, d.h. geregelt, am Kochen gehalten werden.
- Nach 6 Minuten Kochzeit ist die Heizung abzustellen und der Benutzer zum Herausnehmen des Eies aufzufordern, was dieser dann wieder bestätigen soll.
- Vor Beendigung des Programms ist der Benutzer zum Abtrennen der Heizung vom Netz und entsprechender Bestätigung aufzufordern.

  Hinweis: Organisieren Sie Ihr Programm durch den Einsatz von geeigneten Unterprogrammen hierarchisch, damit die Zugriffe auf die Hardware von der eigentlichen Programmlogik entkoppelt werden.

3 Inbetriebnahme der Versuchsanordnung

3.1 Test der Schnittstelle

Dazu wird der für die Prozeßsteuerung verwendete Adapterstecker zunächst abgezogen und durch einen serienmäßigen Schnittstellentester ersetzt, auf dem durch Leuchtdioden die einzelnen Schnittstellensignale angezeigt werden.

- Das Programm soll zuerst auf den Ausgabeleitungen periodisch wechselnde Signalzustände (0,1) erzeugen (blinken). - Für eine optimale Anzeigefrequenz von ca. 1 Hz können Systemfunktionen verwendet werden. -
- Dann soll eine Verbindung zwischen Ausgabe- und Eingabeanschlüssen hergestellt werden (loop back, Schleifentest), um zu prüfen, ob die Ausgabesignale auch wieder gelesen werden können. (z.B. RTS -> CTS, DTR -> DSR beim LINUX-Rechner).

3.2 Anschluß der Prozeßperipherie bestehend aus Steuerung (Relais) für die Heizung (Tauchsieder) und Sensor am Kochtopf. Überprüfung ihrer Funktionsfähigkeit, am besten durch das Schnittstellen-Testprogramm (s. 3.1.)

3.3 Anschluß der Heizung, Einfüllen von Wasser und Einsetzen des Tauchsieders.

4. Durchführung

- Der eigentliche Prozeßaufbau ist nur in einer Ausführung vorhanden und muß von allen LaborteilnehmerInnen im Wechsel benutzt werden. Nehmen Sie die Prozeßperipherie erst dann in Betrieb, wenn Sie sicher sind, daß Ihr Programm auch das leistet, was von ihm erwartet wird.
- Nach Programmstart soll das Programm mit Benutzerführung ablaufen.
- Die beobachteten Prozeßzeiten (Regelzyklen) sind zu notieren.
- Eier sind selber mitzubringen (ein Laden ist in der Nähe !).
- Das gekochte Ei ist vor Ort zu verspeisen.

5. Ausarbeitung

- DV-Handbuch mit Programmdokumentation (Quellcode, Struktogramme, Modulpläne, Petri-Netzen)
- Bedienungsanleitung (Anwenderhandbuch) mit Erläuterung durch ein Zustandsübergangsdiagramm und detaillierte Beschreibung der Versuchsanordnung.

Beschreibung der Beobachtungen bei der Durchführung:

Aufheizzeit (in sec):

Regelzyklen (in /sec):

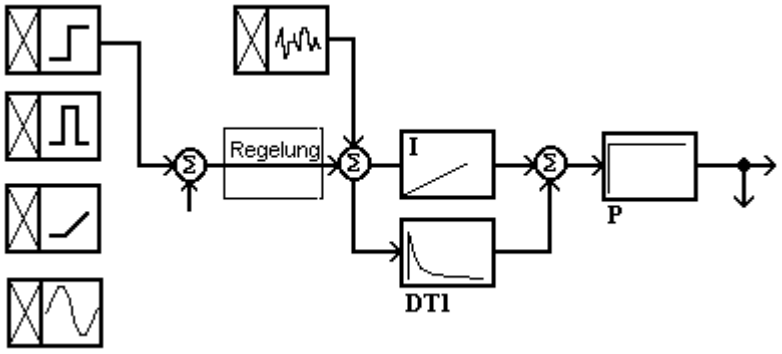
Konsistenz des gekochten Eies: roh ÿ, wachsweich ÿ, hart ÿ

Übung 4b: Modellierung einer Regelstrecke mit RESI

Ziel dieser Aufgabe:
Es ist eine Regelstrecke für ein vorgegebenes System zu modellieren und in der Simulation zu testen.

0. Vorbereitung:

- Strukturen im Wirkungsplan ergänzen.
- Übertragungsfunktion des Systems und der Regelung bestimmen.



1. Grundlagen

Das REgelungs-SImulationsprogramm **RESI** wurde an der FH FFM im Studiengang Ingenieur-Informatik erstellt und enthält noch einige Schwachstellen, die sich aber nur bei Fehlbedienung bemerkbar machen, dann aber oft so dramatisch, daß ein völliges Ausschalten des Rechners nötig wird.
RESI ist frei kopierbar und läuft unter MS-DOS ab Version 5.0, RESI benötigt eine Gleitkomma-Coprozessor und mindestens eine EGA-Graphikausgabe !!!
RESI enthält eine graphische Benutzeroberfläche mit Symbolen für

- Übertragungsglieder (parametrierbare)
- Signalquellen
- Bedienung (intuitiv bedienbar, daher keine ausführliche Bedienungsanleitung)

Folgende Übertragungselemente werden im vorliegenden Versuch eingesetzt:

Typ	Übertragungsfunktion G(s)
P	K_P
I	K_I / s
D	$K_D * s$
DT_1	$K_D * s / (1 + s * T_1)$
PT_2	$K_P / (1 + 2d * s * T_2 + (s * T_2)^2)$

2. Inbetriebnahme

Das Simulationsprogramm RESI läuft unter MS-DOS !!
Zum Booten von DOS ist der Rechner entweder durch RESET oder durch Einschalten zu Booten. Dabei ist der Standard-Bootvorgang durch <ESC> zu unterbrechen und an der Aufforderung LILO: ist dos einzugeben.
Das Simulationsprogramm befindet sich im Ordner RESI. Es wird mit RESI gestartet.

3. Durchführung

3.1 Es wird das zu regelnde System "Auf4b.sim" geladen.
Es enthält 4 unterschiedliche Testsignalgeneratoren, ein System aus I-Glied und DT1-Glied (RC-Hochpaß), eine eingespeiste stochastische Störung und ein P-Glied als Ausgangspuffer.

3.2 Das Verhalten des Systems ist durch Anschalten geeigneter Testfunktionen zu testen.
Die erhaltenen Systemantworten sind auszudrucken oder abzuzeichnen.

3.3 Es ist eine Regelung so zu realisieren, daß die Ausgangsspannung einem vorgegebenen Signalverlauf möglichst getreu folgt.

3.4 Das Verhalten der Regelung ist durch Anschalten der Sprung- und der Impulsfunktion ohne und mit Störsignal zu testen. Die erhaltenen Systemantworten sind auszudrucken oder abzuzeichnen.

4. Erstellung der Regelung

4.1 Die im Regelkreis notwendigen Glieder und deren Anordnung sind aus den (Laplace-transformierten) Übertragungsfunktionen der Systemglieder analytisch zu bestimmen und anzupassen.

4.2 An Stelle des in 4.1 analytisch bestimmten Regelglieds ist ein PID-Glied einzusetzen, dessen Parameter empirisch bestimmt werden müssen. Dessen Regelverhalten ist mit dem aus 4.1 zu vergleichen.

5. Ausarbeitung

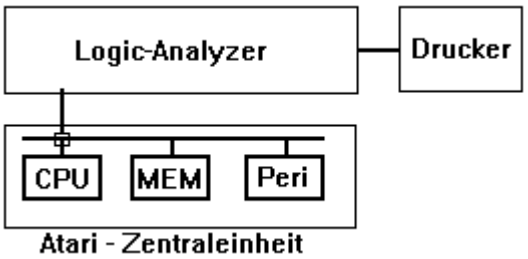
- Beschreibung des zu regelnden Systems analytisch und anhand der Signalantworten der 4 Signalquellen.
- Beschreibung der Regelung analytisch und anhand der Signalantworten aus Rechteck-Sprungfunktion und Rechteck-Impulsfunktion.
- Vergleich der theoretischen und der tatsächlichen Regelabweichungen aus 4.1
- Vergleich des Regelverhaltens aus 4.1 und 4.2.

Übung 4c: Beobachtungen an der Busschnittstelle des Prozessors 68 000 mit einem Logikanalysator

Ziel der Aufgabe: Anhand der Signale auf dem Bus soll
- die Arbeit der CPU beobachtet,
- das Busprotokoll beschrieben und
- der Zeitbedarf von Instruktionen ermittelt werden.

0. Vorbereitung
- Beschaffen Sie sich Unterlagen (Literatur) über die Motorola 68000 CPU, insbesondere über deren Befehlssatz (Anlage D 4) und deren Bussignale und -protokolle.
- Erstellen Sie ein kurzes Assembler-Programm, das in einer Endlosschleife mindestens je einen Befehl mit Schreib- und mit Lesezugriff auf den Arbeitsspeicher enthält. Füllen Sie es mit NOPs, damit ein etwas längerer Programmablauf entsteht.

1. Grundlagen
An die Busschnittstelle eines Prozessors wird ein Logikanalysator (PM 3585) angeschlossen, mit dem die Bussignale aufgezeichnet und auch interpretiert (disassembliert) werden können.
Die Benutzerschnittstelle (s.w.u.) besteht aus Eingabemöglichkeiten über eine Tastatur mit Funktions- und alphanumerischen Tasten, über einen Drehknopf und über eine Maus, einer Bildschirmanzeige und einem Drucker für Hardcopies der Bildschirmanzeige.



2. Inbetriebnahme der Versuchsanordnung:
2.1 Anschalten des Logikanalysators
2.2 Anschluß des Meßkopfs an die Atari-CPU
2.3 Starten des Atari-Rechners
erfolgen durch die Betreuer.
2.4 Erstellen eines eigenen Assembler-Programms
Der Atari (unter dem Betriebsystem TOS) besitzt eine graphische Benutzeroberfläche (ähnlich Windows); auf seiner Systemdiskette befindet sich ein interaktiver Assembler, der sich im Ordner "Omicron-Assembler" befindet und dort aufgerufen wird. Er besitzt einen integrierten Editor zur Programmerstellung, mit dem auch ein Listing des Maschinenprogramms ausgegeben werden kann.

3. Durchführung der Messungen
Nach der Inbetriebnahme befindet sich der Rechner in einer längeren Abfrageschleife, deren Ablauf auf dem Analysator als Impulsdiagramm, als Zustandsliste oder als Disassemblerliste zu beobachten ist.

Bestimmen Sie:
3.1 Die Länge der kürzesten und die der längsten Instruktion in der Abfrageschleife (in µs); zu welchen Instruktionen gehören sie ?
3.2 Die Längen von 2 weiteren, von Ihnen gewählten Instruktionen mit jeweils 1 bzw. 2 Operanden.
- Messen Sie die zeitliche Dauer dieser Instruktionen.
- Bestimmen Sie die Anzahl der Buszugriffe dieser Instruktionen.
- Zeichnen Sie die zugehörigen Impulsdiagramme auf oder machen Sie eine Hardcopy davon, die Sie geeignet interpretieren.
- Bestimmen Sie hieraus die Latenzzeiten des Arbeitsspeichers bei Lese- und bei Schreibzugriffen.

Da keine weiteren Vorkehrungen getroffen werden, um in das Betriebssystem zurückkehren zu können, muß der Rechner nach Beendigung des Programms neu gestartet werden (RESET-Taste).

3.3 Bestimmen Sie den Adreßbereich und die Gesamtlänge der Endlosschleife und stellen Sie den Ablauf als A-t- Diagramm graphisch auf dem Logikanalysator dar. Machen Sie sich eine Hardcopy hiervon.

Hinweise zur Durchführung (vgl. auch E.7 Der Logikanalysator PM 3585)
Betrachten Sie zunächst die Disassemblerlisten auf sinnvolle Instruktionen (Der Einsatzpunkt des Disassemblers kann erst nach mehreren Buszyklen erfolgen ! Warum wohl ?)
Suche Sie dann im Impuls-Diagramm (State Kurve) die entsprechende Stelle heraus und stellen Sie sie in einem geeigneten Maßstab dar. Vergleichen Sie die Kurve mit der entsprechenden Liste.
Vermessen Sie die zu den Instruktionen Ihres Testprogramms gehörenden Buszyklen möglichst genau.
Variieren Sie die verschiedenen Adressierungsarten (Register, immediate, absolut, relativ, deferred, indiziert).

4. Ausarbeitung

- Darstellung der Instruktionen gemäß Arbeitsblatt
- Assemblerlisting und Maschinenprogramm Ihres Testprogramms
- A-t-Diagramm Ihres Testprogramms mit Zuordnung zu den einzelnen Instruktionen
- Angaben zur Länge der Testschleife: Anzahl der Instruktion, Länge in µs, Anzahl der Buszugriffe
- die Latenzzeiten der einzelnen Signale (Adresse, Daten, R/W, DTACK, AS) und interpretieren Sie sie.

Tragen Sie Ihre Ergebnisse in die Aufgabenblätter ein.

D.5 Arbeitsblatt 1 zur Aufgabe 4c: Busprotokolle

Namen: ..... Datum: .....
.....
..... Rechner: Atari 1024, CPU: M68000

Kürzeste Instruktion:

Code: Länge (in µs): Anzahl Buszugriffe:

Timing-Diagramm:

ADRESS-BUS

DATA - BUS

AS(n)

R/W(n)

FC(2_0) -BUS

<------- FETCH ----------><--------Operand ? ---->

Längste Instruktion:

Code: Länge (in µs): Anzahl Buszugriffe:

Timing-Diagramm:

ADRESS-BUS

FC(2_0) -BUS

DATA - BUS

AS(n)

R/W(n)

<------- FETCH ----------><--------Operanden ---->

1-Operanden Instruktion:

Code: Länge (in µs): Anzahl Buszugriffe:

Timing-Diagramm:

ADRESS-BUS

FC(2_0) -BUS

DATA - BUS

AS(n)

R/W(n)

<------- FETCH ----------><--------Operand 1 ---->

2-Operanden Instruktion:

Code: Länge (in µs): Anzahl Buszugriffe:

Timing-Diagramm:

ADRESS-BUS

FC(2_0) -BUS

DATA - BUS

AS(n)

R/W(n)

<------- FETCH ----------><--------Operand 1 ----------><-------- Operand 2 ---->

D.5 Arbeitsblatt 2 zur Aufgabe 4c: Busprotokolle

Namen: ..... Datum: .....
.....
..... Rechner: Atari 1024, CPU: M68000

A-t-Diagramm des Testprogramms:

Ergebnisse:

Länge der Schleife: ..... µs NOP-Instruktion: ..... µs

Länge eines READ-Zyklus: ..... µs WRITE-Zyklus: ..... µs

Zeitabstand (gegen Signal AS(n))

FC ..... (ns) ADRESS ..... (ns)

R/W ..... (ns)

DATA (read) ..... (ns) DT-ACK(n) ..... (ns)

DATA (write) ..... (ns) DT-ACK(n) ..... (ns)

Zeitliche Schwankungen der Signale (Jitter)

ADRESS ..... (ns) DATA (read) ..... (ns)

Interpretationen:

Bussignale des MC 68 000

Adress(1_23) Wort-Adressen
Data(0_15) Daten

Steuerung:

AS(n) Adress Strobe (neg)
UDS(n) Upper Data Strobe (neg)
LDS(n) Lower Data Strobe (neg)
R/W(n) Read / Write (neg)
DTACK(N) Data Acknowledge (neg)

BR(n) Bus Request (neg)
BG(n) Bus Grant (neg)
BGACK(n) Bus Grant Acknowledge (neg)
CLK Clock (8 MHz)

FC2_0 Function Control
`0 0 0` not defined
`0 0 1` Daten im User Mode
`0 1 0` Instruktion im User Mode
`1 0 1` Daten im Kernel Mode
`1 1 0` Instruktion im Kernel Mode
`1 1 1` Interrupt Grant

IPL2_0(n) Interrupt Priority Level
HALT
RESET
BERR Bus Error

Übung 5a: Programmierung, Test und Eichung eines digitalen Photometers

Ziel dieser Aufgabe:

Mit Hilfe der Analogeingabe eines Prozeßrechners sind die Signale einer Solarzelle zu erfassen und als digitale Helligkeitswerte on-line auf dem Bildschirm und off-line als Meßkurve darzustellen.

0. Vorbereitung:

- Überlegen Sie sich, welche Algorithmen Sie verwenden wollen, um die Eichkurve zu erstellen.
- Bereiten Sie ein Programm (Struktogramm) vor, das geeignet strukturiert ist und flexibel an die vorliegenden Gegebenheiten angepaßt werden kann.

1. Grundlagen

a) Die Kennlinie des Photoelements ist weitgehend linear, genauer gesagt ist der Kurzschlußstrom I bis zu einer bestimmten Grenze proportional zur Lichtintensität L (dies würde ein Meßinstrument mit 0Ω erfordern)

$I = c*L$ (z.B. mit c = 50 nA / lux).

Wird aber die Leerlaufspannung U der Photozelle gemessen, findet man eine kompliziertere Abhängigkeit:

$U = U_0 * \ln (1 + L/L_0)$ (z.B. mit U_0 = 60 mV, L_0 = 2 lux)

Diese Funktion ist sehr unhandlich (probieren Sie es).

Für genügend große Helligkeit ($L > 10\ L_0$) gilt näherungsweise

$U = U_0 * \ln(L / L_0)$

Tatsächlich wird die Photozelle parallel zu einem kleinen Lastwiderstand (500 Ω) betrieben, der weder einen Kurzschluß noch einen Leerlauf darstellt, so daß sich eine noch komplexere Abhängigkeit ergibt.
Zur Eichung muß eine Näherungsfunktion benutzt werden, wie sie weiter unten beschrieben wird.

b) Die Zeitauflösung eines Photoelements ist recht gut, sie liegt bei 0.1 ms oder darunter. Damit können zeitabhängige Vorgänge erfaßt werden, denen unser Auge nicht folgen kann, z.B. das Flimmern eines Computer-Bildschirms oder einer Lampe. Bei Intensitätsmessungen an wechselstrombetriebenen Lichtquellen, bei denen das erfaßte Signal zeitabhängig ist, kann es zu erheblichen Fehlmessungen kommen, wenn das Meßgerät keine Mittelwertbildung durchführt oder nicht vollständig über ganze Perioden gemittelt wird. Es ist deshalb notwendig, im Rechner über längere Meßreihen zu mitteln. Dazu wird empfohlen, ca. 1/2 Sekunde lang mit einer Auflösung von ca 1/2 ms zu messen und in eine Datei abzuspeichern.

Für die on-line Darstellung der mittleren Lichtintensität ist der Mittelwert zu bilden, mittels der Eichung umzurechnen und auszugeben. Die Erfassung kann freilaufend oder auf Anforderung (Tastatureingabe) erfolgen.

Für die zeitliche Darstellung des Intensitätsverlaufs kann eine Graphikbibliothek CSLIB verwendet werden (s. Anhang E. 9). Optional kann eine Schnelle Fourier Transformation (FFT) der Meßreihen eingebaut und das Frequenzspektrum dargestellt werden (s. Anhang E.10).

c) Der Ablauf des Programms ist zweckmäßigerweise in folgende Schritte zu gliedern und zu testen:

- Erfassung der Eichpunkte: der zu einem bestimmten bekannten Lichtwert x zugehörige Meßwert y wird erfaßt, der Wert x wird dabei über die Tastatur eingegeben.
- Darstellung der Eichpunkte als Zahlenwerte (Liste) auf dem Terminal
- Berechnung einer Eichkurve zur Berechnung von Eingabewert x auf Meßwert y nach einem Ausgleichsverfahren (s.w.u.)
  - Darstellung der Parameter der Eichkurve als Liste auf dem Terminal
  - Darstellung der Eichkurve als Graphik auf dem Terminal
- Darstellung des aktuellen Prozeßwerts als Mittelwert über genügend viele Meßwerte auf dem Bildschirm als Anzeigeinstrument
- Darstellung einer Meßreihe als Meßkurve auf dem Bildschirm (eventuell ohne Umrechnung).
- Darstellung des Frequenzspektrums einer Meßreihe als Kurve auf dem Bildschirm.

d) Eichkurven

Zur Umrechnung eines Eingabewerts x (Signalspannung) auf den zugehörigen Meßwert y (physikalische Größe) eines Meßglieds wird stets eine Umrechnung benötigt. Im einfachsten Fall ist sie linear, d.h. die Kennlinie (Übertragungsfunktion des Meßglieds) ist $y = c * x$ bzw. die Eichkurve $x = y / c$.
Im allgemeinen werden kompliziertere Funktionen benötigt, $y = g(x)$ und $x = f(y)$, die oft nicht analytisch geschlossen dargestellt oder umgerechnet werden können.
Dann versucht man Näherungsfunktionen anzuwenden, die im einfachsten Fall wieder linear sind.

Ein Verfahren zur Linearisierung ist die bekannte Taylor-Entwicklung um einen Fixpunkt x0:

$$f(x_0 - x) = f(x_0) + x*f'(x_0) + x^2*f''(x_0)/2 + ... + x^n*f^{(n)}(x_0)/n!$$

die nach dem 1. Glied abgebrochen werden kann, falls das 2., quadratische Glied $f''(x_0)$ genügend klein ist.

Für die Erstellung einer linearen Funktion $f(x) = y - a + b * x$ aus n Meßwerten (x_i, y_i) kann man folgendes Verfahren anwenden:

Zuerst berechnet man den Korrelationskoeffizienten r

$$r = S_{xy} / SQRT(S_{xx} * S_{yy}) \quad \text{mit} \quad S_{xx} = \Sigma x_i^2 - (\Sigma x_i)^2 \; ; \quad S_{yy} = \Sigma y_i^2 - (\Sigma y_i)^2/n \; ; \quad S_{xx} = \Sigma x_i y_i - (\Sigma x_i y_i)/n$$

Den Wert r prüft man, ob er genügend gut bei 1 liegt; ist dies der Fall, so berechnet man dann die Koeffizienten

$$b = S_{xy} / S_{xx} \quad \text{und} \quad a = (\Sigma y_i - b*\Sigma x_i)/n$$

Falls kein linearer Zusammenhang besteht, für die nichtlineare Übertragungsfunktion g jedoch eine Umkehrfunktion f angegeben werden kann, können deren Parameter aus einer entsprechenden Anzahl von Meßwertpaaren (x_i, y_i), nach Auflösung von meist nichtlinearen Gleichungssystemen, bestimmt werden.
Wenn deren analytische Auflösung nicht möglich ist, müssen dafür numerische Verfahren eingesetzt werden.

2. Inbetriebnahme der Versuchsanordnung

Photoelement, Eichlampe und Oszilloskop werden von den Betreuern bereitgestellt.

3. Erstellung des Meßprogramms

Das Meßprogramm ist in einer höheren Programmiersprache (vorzugsweise "C") zu erstellen.
Für den Zugriff auf die Prozeßperipherie sind die Bibliotheken 'libvaio.a' (für Analoge Ein/Ausgabe) und 'libvdio.a' (für Digitale Ein/Ausgabe) vorhanden, die bei der Programmerstellung in das Anwenderprogramm mit eingebunden werden müssen. Die wichtigsten Funktionen sind in den Anlagen E.5 und E.6 zusammen gefaßt

Es empfiehlt sich, den Synchronbetrieb zu wählen und folgende Werte zu benutzen:

vstart = vend = 1 (A/D-Kanal 1), vadsrv=17 (CONTINUIERLICH)
ad_frames = 0x7FFF = 32767 (o.ä.) ad_buffers = 2 (oder mehr)
dastart = daend = 1 (D/A-Kanal 1), vdasrv=1, da_frames = da_buffers = 1
(tatsächlich werden gar keine D/A-Kanäle verwendet)
cnvtime = 500 000 (ergibt eine Abtastfrequenz von 2 kHz)
ad_frames, ad_buffers und cnvtime werden mit den tatsächlichen Werten der Hardware abgeglichen.
Es empfiehlt sich, die Meßwerte unverzüglich in Dateien abzuspeichern und später von dort zu lesen.
Für die graphische Darstellung von Kurven auf dem Bildschirm kann eine bereitgestellte Graphik-Bibliothek CSLIB benutzt werden, die in der Anlage E.9 beschrieben ist.
Auf jeden Fall soll Ihr Programm in einem Vorspann einen Test aller Ein- und Ausgabe-Kanäle vorsehen!

4. Durchführung

4.1 Eichung der Solarzelle mit einer Glühlampe

Die Eichung erfolgt bei folgenden Lichtintensitäten:

- Dunkelwert (Abdeckung mit der Hand L = 0 lux), der Eingangswert muß 0 sein (nachprüfen), sonst muß dieser Dunkelwert von allen folgenden Werten abgezogen werden.
- direkte Beleuchtung mit Tischlampe (40 W matt) aus einer Entfernung l vom Lampenkolben:
  600 lux : 40 cm
  1800 lux : 20 cm
  6000 lux : 10 cm (welche physikalische Gesetzmäßigkeit steht dahinter ?)

Bei größeren Abständen empfiehlt sich eine Abschirmung von Fremdlicht durch ein Papprohr entsprechender Länge.

4.2 Messung der Lichtverhältnisse im Labor

- Messungen der Lichtintensitäten am Fenster, in der Raummitte, an der Tür
- Darstellung der zeitlichen Verläufe an einer Leuchtstofflampe oder an einem Bildschirm mit einer Auflösung von 1 kHz
- Zur Kontrolle soll mit einem Oszilloskop parallel zum Meßeingang des Rechners das Signal des Photoelements beobachtet und von Hand aufgezeichnet werden.

5. Ausarbeitung

- DV-Handbuch mit Programmdokumentation (Quellcode, Struktogrammen, Modulpläne)
- Bedienungsanleitung (Anwenderhandbuch und Versuchsaufbau)
- Darstellung der Beobachtungen bei der Durchführung auf dem Arbeitsblatt

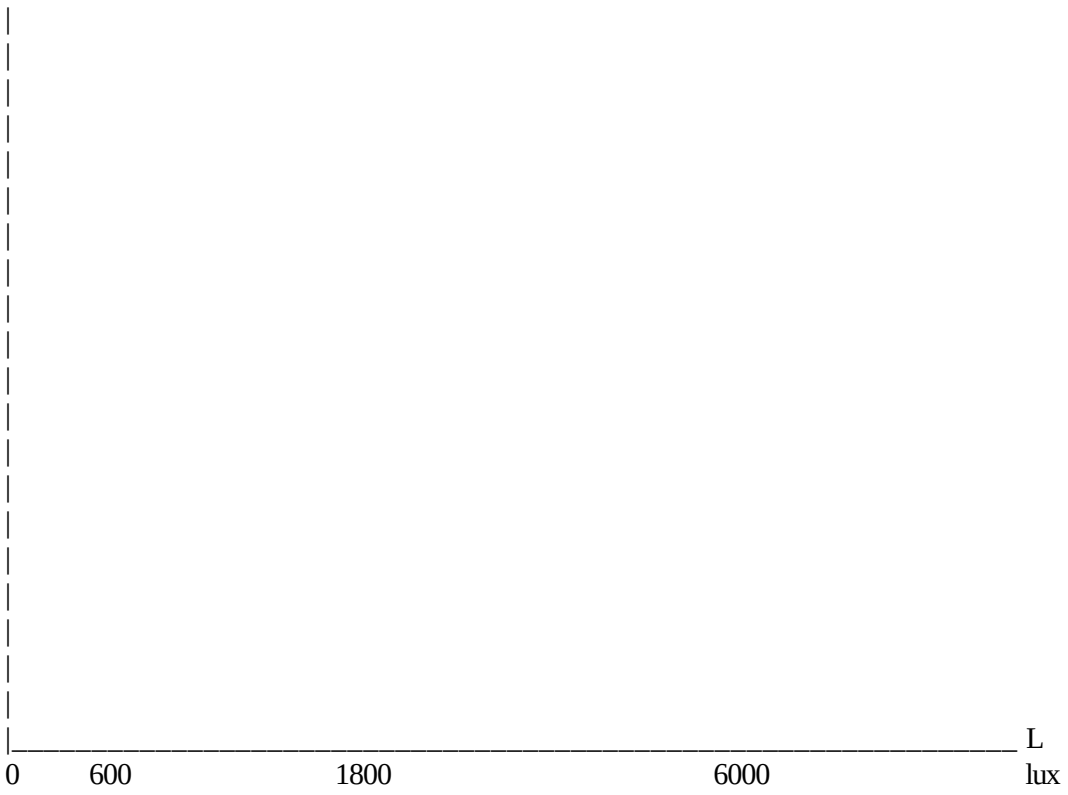
Arbeitsblatt zu Übung 5a: Test und Eichung eines digitalen Photometers

Namen: ..... Datum: .....
Rechner: ... Sensor: .....

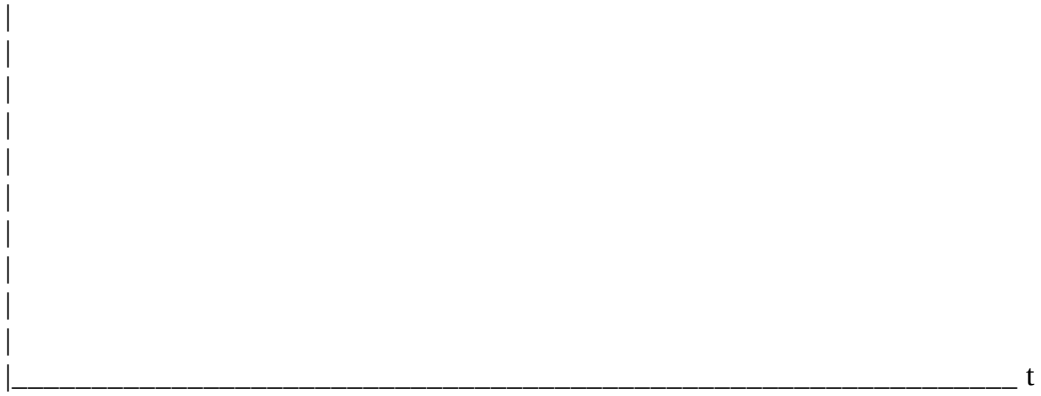
Eichwerte:

Intensität	Abstand	Meßwert
600 lux :	40 cm	. . . . .
1800 lux :	20 cm	. . . . .
6000 lux :	10 cm	. . . . .

Eichkurve:



Eichfunktion: y = a * x + b
Eichfaktoren: a = ... b = ... (Maßeinheiten)



Skizze der Meßkurve am Oszilloskop bei Beobachtung
- einer Leuchtstofflampe oder
- eines Computerbildschirms .....

Übung 5b: Programmierung, Test und Eichung eines digitalen Thermometers

Ziel dieser Aufgabe:
Es sollen mit einem NTC-Widerstand Temperaturen erfaßt und angezeigt werden.
Mit Hilfe der Analogeingabe eines Prozeßrechners sind die Signale einer Thermowiderstands (NTC) zu erfassen und als digitale Temperaturwerte on-line auf dem Bildschirm und off-line als Meßkurve darzustellen.

0. Vorbereitung:

- Überlegen Sie sich, welche Algorithmen Sie verwenden wollen, um die Eichkurve zu erstellen.
- Bereiten Sie ein Programm (Struktogramm) vor, das geeignet strukturiert ist und flexibel an die vorliegenden Gegebenheiten angepaßt werden kann.

1. Grundlagen

a) Der Temperatursensor (NTC-Widerstand) hat eine relativ große Trägheit, seine Einstellzeit beträgt mindestens 0.1 Sekunden (100 ms). Da Temperaturänderungen in der Regel langsam erfolgen, ist dies auch ausreichend. Dennoch sollten bei der Eichung Meßreihen mit geeigneter Zeitauflösung erfaßt und ausgewertet werden, um die Einstellzeit zu bestimmen und um irgendwelche Störungen zu erkennen.

b) Die Kennlinie eines NTC-Widerstands ist ausgesprochen nichtlinear,
näherungsweise gilt (für die absolute Temperatur T in Kelvin):
$R(T) = R_0 * \exp(T_0/T)$ (z.B. mit $R_0 = 0.04$ Ohm, $T_0 = 2500$ K)

Damit hat ein NTC-Widerstand einen negativen Temperaturkoeffizienten (Negative Temperature Coefficient), d.h. der Widerstand sinkt mit steigender Temperatur.
Da die Kennlinie nur in kleinen Bereichen näherungsweise als linear betrachtet werden kann, ist sie mit genügend vielen Eichpunkten zu erfassen und die Eichkurve zu bestimmen.

Die Eichung erfolgt am besten bei folgenden Temperaturen:

20 °C	: Zimmertemperatur
0 °C	: Eiswasser
100 °C	: Kochendes Wasser (vom Eierkochen, Laborversuch 1)
50 °C	: Mischwasser (50:50)

Für die Erstellung einer linearen Funktion $f(x) = y - a + b * x$ aus n Meßwerten (x_i, y_i) kann man folgendes Verfahren anwenden:
Zuerst berechnet man den Korrelationskoeffizienten r

$$r = S_{xy} / SQRT(S_{xx} * S_{yy}) \quad \text{mit} \quad S_{xx} = \Sigma x_i^2 - (\Sigma x_i)^2 \; ; \quad S_{yy} = \Sigma y_i^2 - (\Sigma y_i)^2/n \; ; \quad S_{xx} = \Sigma x_i y_i - (\Sigma x_i y_i)/n$$

Den Wert r prüft man, ob er genügend gut bei 1 liegt; ist dies der Fall, so berechnet man dann die Koeffizienten

$$b = S_{xy} / S_{xx} \quad \text{und} \quad a = (\Sigma y_i - b*\Sigma x_i)/n$$

Falls kein linearer Zusammenhang besteht, für die nichtlineare Übertragungsfunktion g jedoch eine Umkehrfunktion f angegeben werden kann, können deren Parameter aus einer entsprechenden Anzahl von Meßwertpaaren (xi, yi), nach Auflösung von meist nichtlinearen Gleichungssystemen, bestimmt werden.
Dieses Verfahren empfiehlt sich für die Bestimmung der Einstellzeit t_z, denn die Einstellung einer Meßgröße läßt sich fast immer durch folgende Funktion darstellen:

$$y(t) = Y_0 (1 - \exp(-t/t_z))$$

c) Der Ablauf des Programms ist zweckmäßigerweise in folgende Schritte zu gliedern und zu testen:

- Erfassung der Eichpunkte:
  der zu einem bestimmten bekannten Temperaturwert x zugehörige Meßwert y wird erfaßt, der Wert x wird dabei über die Tastatur eingegeben.
  Es empfiehlt sich, nach getaner Eichung diese Werte in einer Datei abzuspeichern, um die aufwendige Temperatureichung nicht jedesmal wiederholen zu müssen.
- Bestimmung und Anzeige der Einstellzeiten
- Darstellung der Eichpunkte in einem Koordinatenkreuz auf dem Arbeitsblatt
- Berechnung einer Eichkurve zur Berechnung von Eingabewert auf Meßwert nach einem Ausgleichsverfahren (s.w.u.)
- Darstellung der Eichkurve
- Darstellung des aktuellen Meßwerts auf dem Terminal als Anzeigeinstrument

2. Inbetriebnahme der Versuchsanordnung

Temperatursensor, Eis und Kocher werden von den Betreuern bereitgestellt.

3. Erstellung des Meßprogramms

Das Meßprogramm ist in einer höheren Programmiersprache (vorzugsweise "C") zu erstellen.

Für den Zugriff auf die Prozeßperipherie sind die Bibliotheken 'libvaio.a' (für Analoge Ein/Ausgabe) und 'libvdio.a' (für Digitale Ein/Ausgabe) vorhanden, die bei der Programmerstellung in das Anwenderprogramm mit eingebunden werden müssen.:

Die wichtigsten Funktionen sind in den Anlage E5. und E.6 zusammengefaßt

Es empfiehlt sich, den Asynchronbetrieb (modefree) zu wählen und folgende Werte zu benutzen:

vstart = vend = 1 (A/D-Kanal 1), vadsrv=2 (CORRIGIERT)

dastart=daend=1 (D/A-Kanal 1), vdasrv=1 (zur Ausgabe einer festen Referenzspannung für den NTC)

cnvtime = 1 000 000 (ergibt eine Abtastfrequenz von 1 kHz, wird mit dem tatsächlichen Wert der Hardware abgeglichen).

Bedenken Sie, daß der NTC-Widerstand ein passives Bauelement ist und selber keine Spannungen liefert, statt dessen muß eine Stromquelle in Form einer Ausgabespannung und eines Vorwiderstands emuliert werden. Man überlege sich, welche Auswirkung dies auf die Eichung hat.

Auf jeden Fall soll Ihr Programm in einem Vorspann einen Test aller Ein- und Ausgabe-Kanäle vorsehen !

4. Durchführung

- Erfassung der Eichpunkte:
- Darstellung der Eichpunkte in einem Koordinatenkreuz auf dem Arbeitsblatt
- Berechnung einer Eichkurve zur Berechnung von Eingabewert auf Meßwert nach einem Ausgleichs-verfahren (s.w.o.)
- Darstellung der Eichkurve auf dem Arbeitsblatt
- Darstellung des aktuellen Meßwerts auf dem Terminal als Anzeigeinstrument
- Messung der Raumtemperatur am Fenster, in der Raummitte und an der Tür
- Messung der Körpertemperaturen der Versuchsteilnehmer

5. Ausarbeitung

- DV-Handbuch mit Programmdokumentation (Quellcode, Struktogrammen, Modulpläne)
- Bedienungsanleitung (Anwenderhandbuch und Versuchsaufbau)
- Beschreibung der Beobachtungen bei der Durchführung an Hand des Arbeitsblatts.

Arbeitsblatt zur Übung 5b: Programmierung, Test und Eichung eines digitalen Thermometers

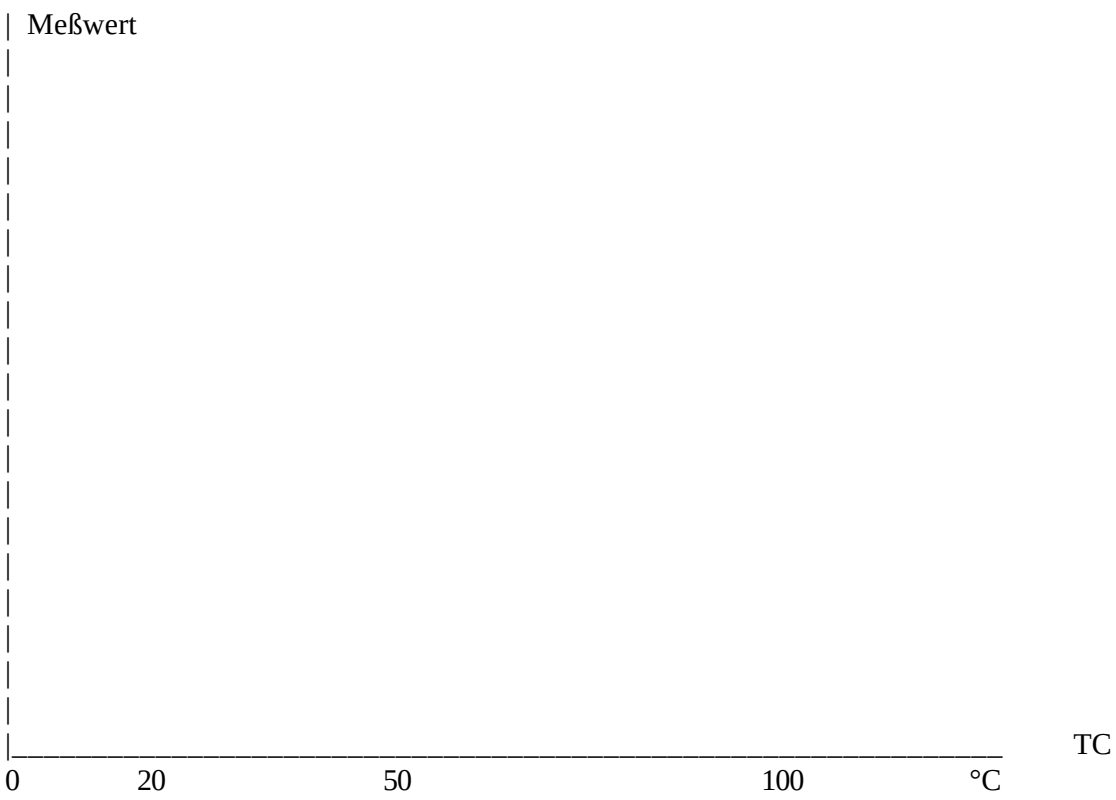
Namen: ..... Datum: .....

Rechner: ... Sensor: .....

Eichwerte:

Intensität	**Abstand**	**Meßwert**
Zimmertemperatur	20 °C :	. . . . .
Eiswasser	0 °C :	. . . . .
Kochendes Wasser	100 °C :	. . . . .
Mischwasser (50:50)	50 °C :	. . . . .

Eichkurve: Einstellzeit: ..... s



Eichfunktion: y = a * x + b
Eichfaktoren: a = ... b = ... (Maßeinheiten)

Messung der Raumtemperatur
am Fenster . . . . . °C
in der Raummitte . . . . . °C
an der Tür . . . . . °C

Messung der Körpertemperaturen der Versuchsteilnehmer:
1. Teilnehmer(in): . . . . . °C
2. Teilnehmer(in): . . . . . °C

Übung 5c: Programmierung und Test für die Erfassung von Schallereignissen

Ziel dieser Aufgabe:

Mit Hilfe der Analogeingabe eines Prozeßrechners sind die Signale eines Mikrofons zu erfassen und als Zeitverlauf und als Frequenzspektrum on-line auf dem Bildschirm darzustellen.

0. Vorbereitung:

- Überlegen Sie sich, welche Algorithmen Sie verwenden wollen, um das Frequenzspektrum zu erstellen und auszuwerten.
- Bereiten Sie ein Programm (Struktogramm) vor, das geeignet strukturiert ist und flexibel an die vorliegenden Gegebenheiten angepaßt werden kann.

1. Grundlagen

Schallereignisse bestehen aus akustischen Wellen im Frequenzbereich von ca. 20 Hz bis 20 kHz.

Um solche Analogwerte digital aufzunehmen, müssen sie nach Shannon mit mindestens der doppelten Frequenz also 40 kHz abgetastet werden. Aus Speicherökonomie ist hier eine Beschränkung auf ca. 32 kHz vorzunehmen. Ebenso ist eine zeitliche Begrenzung auf ca. 1 bis 2 Sekunden für jedes Schallereignis vorzusehen.

2. Inbetriebnahme der Versuchsanordnung

Mikrofone werden von den Betreuern bereitgestellt.

3. Erstellung des Meßprogramms

Das Meßprogramm ist in einer höheren Programmiersprache (vorzugsweise "C") zu erstellen.

Für den Zugriff auf die Prozeßperipherie sind die Bibliotheken 'libvaio.a' (für Analoge Ein/Ausgabe) und 'libvdio.a' (für Digitale Ein/Ausgabe) vorhanden, die bei der Programmerstellung in das Anwenderprogramm mit eingebunden werden müssen.

Die wichtigsten Funktionen sind in den Anlage E.5 und E.6 zusammengefaßt.

Es empfiehlt sich, den Synchronbetrieb zu wählen und folgende Werte zu benutzen:

dastart=daend=1 (D/A-Kanal 1), vdasrv=1 (tatsächlich werden gar keine D/A-Kanäle verwendet)
vstart = vend = 1 (A/D-Kanal 1), vadsrv=17 (CONTINUIERLICH)

ad_frames = 0x7FFF = 32767	ergibt einen Puffer in 1 Sekunde und es reicht	ad_buffers = 2
da_frames = 0x7FFF = 32767	ergibt einen Puffer in 1 Sekunde und es reicht	da_buffers = 1
cnvtime = 31 250	ergibt eine Abtastfrequenz von 32 kHz	

ad_frames, ad_buffers, da_frames, da_buffers und cnvtime werden mit den tatsächlichen Werten der Hardware abgeglichen.

Es empfiehlt sich, die Meßkurven unverzüglich in Dateien abzuspeichern und von dort auszulesen.

Für die graphische Darstellung von Kurven auf dem Bildschirm wird eine graphische Bibliothek CSLIB verwendet, die im Anhang E.9 kurz beschrieben ist.

Für die schnelle Fourier-Transformationen stehen Funktionen aus einer Bibliothek FFTW (Fast Fourier Transform West) zur Verfügung, die beim Binden angegeben werden muß (s. Anhang E. 10). Andere, einfachere Verfahren können aus der Mathematikvorlesung selbst programmiert oder im Internet gefunden werden.

4. Durchführung

- Erfassung der Audiosignale der 5 Vokale (a e i o u) und der 5 Konsonanten k l r s t.
- Erstellung der Fourier-Transformierten und Darstellung als Audio-Spektrum (Graphik oder Liste)
- Ausgabe der Grundfrequenz (Maximum des Frequenzspektrums) und der relativen Amplituden der ersten 5 Oberwellen (als Liste).

5. Ausarbeitung

- DV-Handbuch mit Programmdokumentation (Quellcode, Struktogrammen, Modulpläne)
- Bedienungsanleitung (Anwenderhandbuch)
- Darstellung der Beobachtungen (in einer Tabelle):

  Für jeden der **Vokale** sind zu bestimmen:

  - Frequenz des **Grundtons** in Hz (Maximum des Frequenzspektrums)
  - relative Amplituden der ersten 5 Obertöne (Harmonische)

  Für die **Konsonanten** sind zu bestimmen:

  - Frequenz des Grundtons im letzten Drittel des Tones (Maximum des Frequenzspektrums oder den spektralen Schwerpunkt)
  - Unter- und Obergrenze des Frequenzspektrums (jenseits derer keine Komponenten größer als - 20 dB vorkommen)
  - Einschwingzeit, d.h. die Zeit, ab der die Amplitude des Grundtons zu 50 % erreicht wird (hierfür wäre ein digitales Frequenzfilter sehr nützlich !).

Übung 5d: Programmierung und Test für die Steuerung eines Plotters

Ziel dieser Aufgabe:

Mit Hilfe der Analogausgabe eines Prozeßrechners ist ein XY-Schreiber so zu steuern, daß vorgegebene Kurven und Texte dort dargestellt werden.

0. Vorbereitung:

Bereiten Sie ein Programm (Struktogramm) vor, das geeignet strukturiert ist und flexibel an die vorliegenden Gegebenheiten angepaßt werden kann.

1. Grundlagen

Bedenken Sie, daß die Mechanik eines Schreiber recht träge ist und der Ausgabegeschwindigkeit eines Computers in der Regel nicht folgen kann.
Strukturieren Sie Ihr Programm so, daß es weitgehend unabhängig vom konkreten Ausgabegerät ist und z.B. auch mit einem Oszilloskop als Ausgabegerät verwendet werden kann.

2. Inbetriebnahme der Versuchsanordnung

Der Plotter samt Stiften und Schreiberpapier werden von den Betreuern bereitgestellt.

3. Erstellung des Meßprogramms

Das Meßprogramm ist in einer höheren Programmiersprache (vorzugsweise "C") zu erstellen.

Für den Zugriff auf die Prozeßperipherie sind die Bibliotheken 'libvaio.a' (für Analoge Ein/Ausgabe) und 'libvdio.a' (für Digitale Ein/Ausgabe) vorhanden, die bei der Programmerstellung in das Anwenderprogramm mit eingebunden werden müssen. Die wichtigsten Funktionen sind in den Anlage E5. und E.6 zusammen gefaßt

Es empfiehlt sich, den Asynchronbetrieb (modefree) zu wählen und folgende Werte zu benutzen:

vstart = vend = 1 (A/D-Kanal 1), vadsrv=2 (CORRIGIERT)
dastart= 1, daend=3 (D/A-Kanal 1, 2, 3), vdasrv= 2
cnvtime = 1 000 000 (ergibt eine Abtastfrequenz von 1 kHz, wird mit dem tatsächlichen Wert der Hardware abgeglichen).

Auf jeden Fall soll Ihr Programm in einem Vorspann einen Test aller Ein- und Ausgabe-Kanäle vorsehen !

Für die Darstellung von Objekten, insbesondere für die Textausgabe empfiehlt es sich, 2 Grundfunktionen zu erstellen:

- polyline(anzahl, *feld) Linienzug mit den Eckpunkten in Array "feld"[2][anzahl]
- kreisbogen(x_m, y_m, r, α_{start}, α_{end}) Kreisbogen um den Mittelpunkt (x_m, y_m), mit Radius r und Start- bzw. Endwinkeln α_{start}, α_{end}

die wiederum auf Basisfunktionen aufbauen, wie

- moveto(x,y) Ziehen mit abgehobenem Stift
- drawto(x,y) Ziehen mit gesenktem Stift

4. Durchführung

- Es sind folgende graphische Objekte auf dem DIN A3 X-Y-Plotter darzustellen:
  - **Rechteck** der Größe 30 x 40 cm
  - **Kreis**, der von 3 Seiten des Rechtecks den Abstand von 2 mm hat
  - Darstellung des Kreismittelpunkts als liegendes **Kreuz** (x)
  - ein beliebiger sinnvoller **Text** mit mindestens 6 verschiedenen Buchstaben an beliebiger Stelle.

5. Ausarbeitung

- DV-Handbuch mit Programmdokumentation (Quellcode, Struktogrammen, Modulpläne)
- Bedienungsanleitung (Anwenderhandbuch)
- Plotterausgabe (s.o.)

6. Einstellungen an XY-Schreiber (Kipp & Zonen)

- X-Anschluß an Kanal 1
- Y-Anschluß an Kanal 2
- Z-Anschluß an der Geräterückseite umstecken und mit Kanal 3 verbinden.
- Alle Schiebeschalter müssen in der oberen Stellung stehen
- Empfindlichkeiten in X- und Y-Richtung sind vorher zu bestimmen
- Alle Nullpunktlagen (ZERO) sind auf 0 zu stellen, so daß der Nullpunkt in der linken unteren Ecke ist.

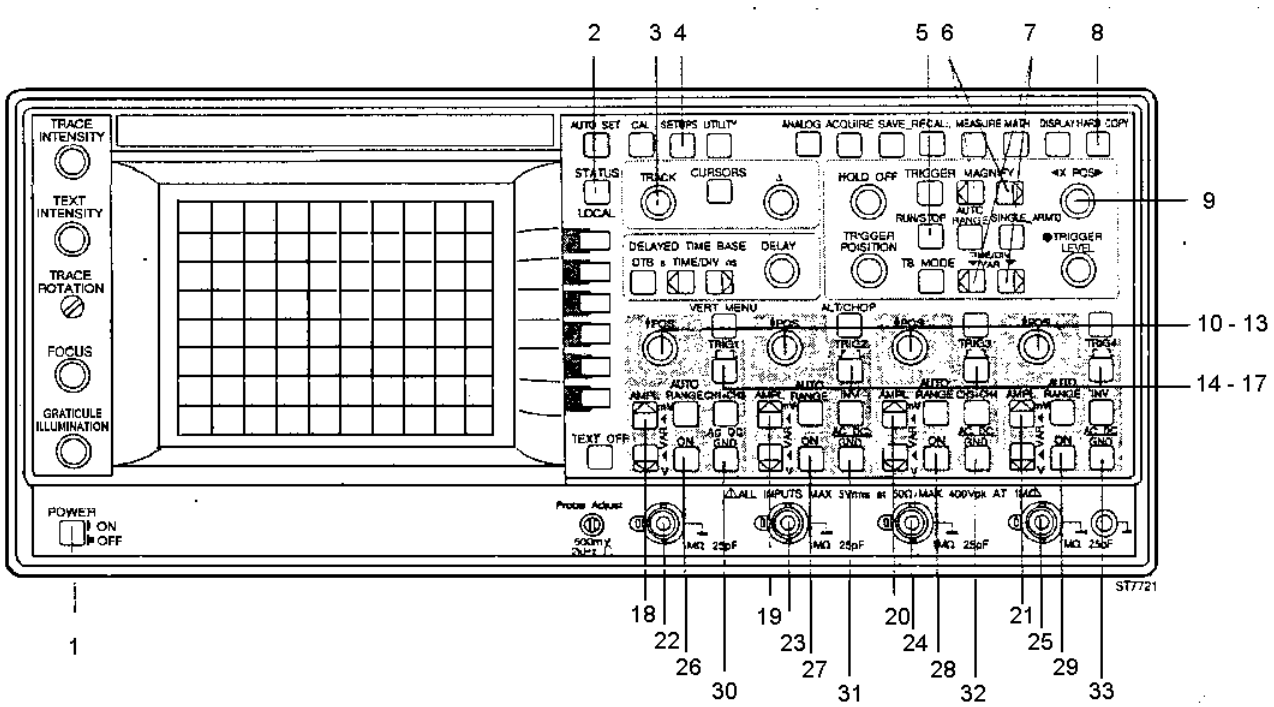
E.1 Schalterstellungen des Philips PM 3382A-Oszilloskops

Eine für die Durchführung der Aufgabe 1.1 passende Voreinstellung kann über das SETUP-Menü jederzeit abgerufen werden:

- **SETUPS** (4) Drücken
- mit **TRACK**-Drehknopf (3) Setup **s1** auswählen
- **RECALL** im Display-Menü wählen
- **SETUPS** (4) drücken.

Für weitere Einstellungen hier die wichtigsten Funktionen:

Nr.	Name	Funktion	Voreinstellung
1	POWER	Netzschalter	
2	STATUS	zeigt die aktuellen Einstellungen an	
5	RUN/STOP	startet und stoppt die Aufzeichnung (Momentaufnahme)	
6	MAGNIFY	vergrößert die aktuelle Anzeige bis 32-fach	
7	TIME/DIV	setzt die Zeitbasis (in sec/cm)	500 ns
8	HARDCOPY	druckt die Anzeige auf dem Drucker aus	
9	X-POS	verschiebt alle Signale horizontal	
10-13	Y-POS	verschiebt das Signal vertikal	
14-17	TRIGx	wählt Kanal x als Triggersignal bzw. Triggern bei steigender o. fallender Flanke	Kanal 1 steigende Flanke
18-21	AMPL	setzt Verstärkung (in Volt pro Kästchen)	0,5V bzw. 5V
22-25	INPUT	Signaleingänge 1 - 4	
26-29	ON	schaltet den Kanal ein/aus	alle Kanäle ein
30-33	AC/DC/GND	Kanalkopplung: Gleich-, Wechselspannung, Erde	DC



E.2 Schalterstellungen des Philips-Oszilloskops PM 3208

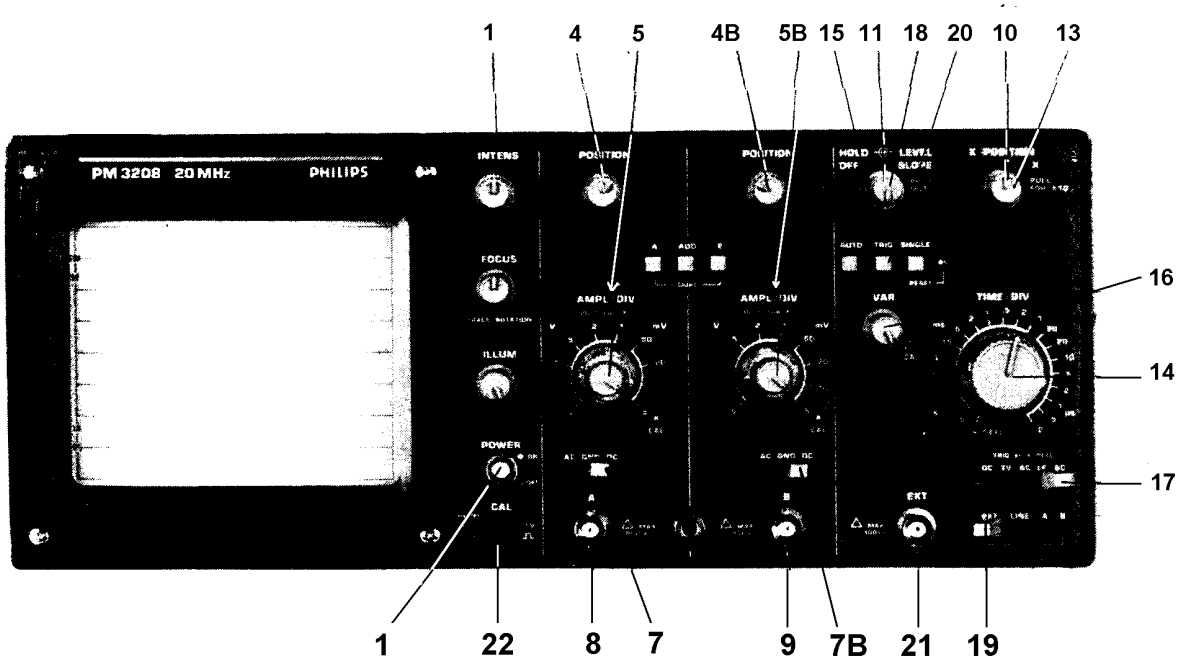
Einstellungen, die ausprobiert werden müssen, sind unterstrichen:

1. Netzschalter + Intensität: Mittelstellung
2. Fokus: nicht verändern.
3. Trace Rotation: nicht verändern
4. Position (A): Mittel- Linie
5. Empfindlichkeit: .5 V/div
6. Eingangs-Schalter: on
7. Spannungs-Schalter: DC
8. Eingang A : Tastkopf mit Abschwächung 1 : 10
9. Eingang B : Tastkopf mit Abschwächung 1 : 10

6B Eingangs-Schalter: on

7B Spannungs-Schalter: s.o.

10. X-Position: Strahlbeginn am linken Rand
11. Trigger-Level: ca. Mitte
12. Zeitbasis: ms/µs
13. Magnification: x1
14. Zeitbasis: ca. 2µs
15. Nachlauf: var
16. Zeitbasis var.: zum Messen auf "cal" (geeicht)
17. Trigger DC
18. Trigger-Slope: +/-
19. Trigger-Quelle: A
20. Trigger-Modus: NORMal
21. Trigger-Anschluß: unbenutzt
22. Eich-Impuls



E.3 Schalterstellungen des GRUNDIG Oszilloskops MO 30

Einstellungen, die verändert oder ausprobiert werden müssen, sind unterstrichen:

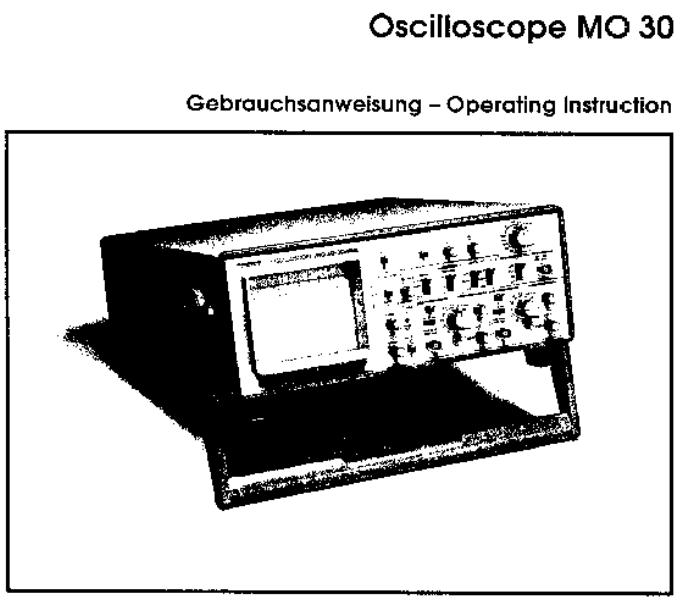
1. Netzschalter
2. Netz-Anzeige
3. Intensität (Mittelstellung !)
4. Fokus: nicht verändern.
5. Eichspannung
6. Trace Rotation: nicht verändern

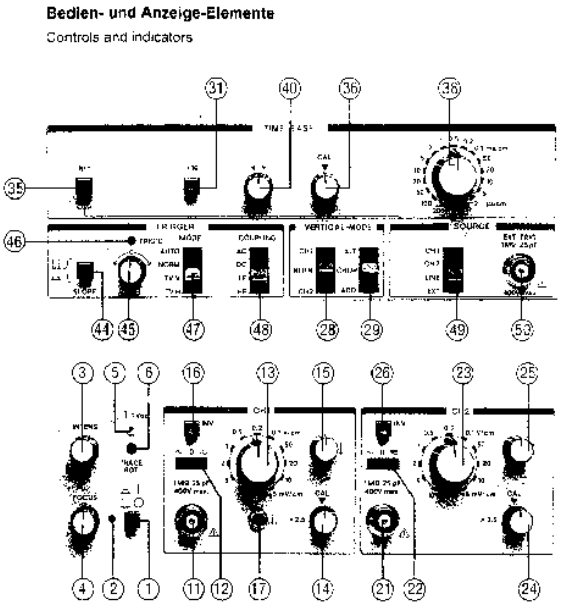
11. Eingang CH1: Tastkopf mit Abschwächung 1 : 10
12. Eingangskopplung: DC ~
13. Empfindlichkeit: .5 V/div
14. Var.: CAL
15. Position (CH1): Mittel- Linie
16. Polarität: nicht invert

21. Eingang CH2: Tastkopf mit Abschwächung 1 : 10
22. Eingangskopplung: DC ~
23. Empfindlichkeit: .5 V/div
24. Var.: CAL
25. Position (CH2): Mittel- Linie
26. Polarität: nicht invert
28. Vertical Mode: BOTH
29. Vertical Mode: CHOP
31. Magnification: x1
35. X/Y: aus

36. Zeitbasis Var.: zum Messen auf "CAL" (geeicht)
38. Zeitbasis: ca. 2µs/cm
40. X-Position: Strahlbeginn am linken Rand

44. Trigger-Slope:
45. Trigger-Level: ca. Mitte (0)
47. Trigger-Modus: AUTO / NORMal
48. Trigger-Kopplung: AC
49. Trigger-Quelle: CH1
50. Trigger-Anschluß: unbenutzt





E.5 Die Digitale In/Out Schnittstelle VDIO32

Die digitale Ein/Ausgabe-Schnittstelle VDIO 32 bietet 32 von einander unabhängige optoentkoppelte Ein- und Ausgänge für digitale Signale von Spannungen zwischen 0 und 24 Volt.
Davon sind 16 fest als Ausgänge geschaltet, die restlichen 16 Leitungen können in Gruppen zu je 4 Bit wahlweise auf Input oder Output geschaltet werden. Gleichgültig wie die Datenrichtung ist, können die extern anliegenden effektiven Pegel der Leitungen jederzeit als Bits ("0" oder "1") gelesen werden.
Die Ausgänge liefern Spannungen von 24 Volt und Ströme bis 300 mA.
Die Eingänge haben eine Schwellenspannung (für logische "0" / "1") zwischen 2.0 und 3.0 Volt

Zur Anbindung von Anwenderprogrammen steht eine Funktionen-Bibliothek "libvdio.a" zur Verfügung, die mit "**libvdio.a**" und mit "**liblynx.a**" gelinkt werden muß.
Benötigte Headerdatei: #include <vdio.h>

Bibliotheksfunktionen - Übersicht

5.1 vdio_open() - Öffnen des Gerätes
5.2 vdio_close() - Deaktivierung, Freigabe der Ressourcen
5.3 vdio_irqptn() - Definiert das Pattern für den Pattern-Match-Interrupt
5.4 vdio_irqon() - Enable Pattern Interrupt. Schaltet den Pattern Interrupt ein
5.5 vdio_irqoff() - Disable Pattern Interrupt. Schaltet den Pattern lnterrupt aus.
5.6 vdio_irqwait() - Warte auf Pattern Interrupt
5.7 vdio_write() - Schreiben 32 Bit breit, mit Maske.
5.8 vdio_read() - Lesen aller 32 Port-Bits
5.9 vdio_writebit() - Schreiben eines einzelnen Port-Bit
5.10 vdio_readbit() - Lesen eines einzelnen Port-Bit
5.11 vdio_readstatus() - Lesen des Fehlerstatus der Ausgangstreiber

Bibliotheksfunktionen

5.1 vdio_open() - Öffnen des Gerätes

int vdio_open("/dev/vdio");

Öffnet das durch den Gerätenamen (/dev/vdio) spezifizierten Gerät, prüft den Installationszustand, erfragt die im Kernel hinterlegten Geräteinformationsdaten (Device Info) und mappt die Register des Gerätes in den Adressraum des Prozesses. Das Mapping wird durch ein sogen. Physical Shared Memory (PSMEM) realisiert. Dazu muß die Anwendung nicht nur mit "libvdio.a", sondern auch mit "liblynx.a" gelinkt werden.
RETURN vdio_open liefert eine Zahl zurück, die für nachfolgende Funktionsaufrufe als "Handle" dient.
Man beachte, dass diese Zahl kein Filedeskriptor ist, sondern ein interner Index der Bibliothek.
Im Fehler fall liefert vdio_open() den Wert -1, weitere Fehlerinformation findet sich ggf. in errno.

EBUSY Treiber bereits geöffnet
errno errno-Codes von open() oder smem_create()

Insbesondere ist es nicht gestattet, ein schon geöffnetes Gerät im gleichen Programm ein zweites Mal zu öffnen, ohne es vorher zu schließen.

5.2 vdio_close() - Deaktivierung, Freigabe der Ressourcen

int vdio_close(int vd);

Ruft die close-Funktion des Treibers auf, blockiert alle lnterrupts des Geräts und deallokiert alle durch die Bibliothek beanspruchten Ressourcen, entfernt die Register der VME-DPIO32-Hardware aus der Memory-Map des Prozesses (SM_DETACH) und versucht schließlich das zugehörige PSMEM zu löschen.
int vd - der von vdio_open() erhaltene "Handle"

RETURN Im Fehlerfall wird -1 geliefert mir entspr. errno-Code, sonst 0.
EBADF vd ist kein gültiger Handle von vdio_open().
errno errno-Codes von close().

CAVEAT
1. vdio_close() wird *nicht* automatisch beim exit() aufgerufen.
2. Das Löschen des PSMEM mißlingt, wenn zu diesem Zeitpunkt noch andere Prozesse Zugriff darauf haben. Dies gilt nicht als Fehler.

5.3 vdio_irqptn() - Set Interrupt Pattern
int vdio_irqptn(int vd, enum dpio32port port, int pattern); Definiert das Muster für den Pattern-Match-Interrupt

5.4 vdio_irqon() - Enable Pattern Interrupt
int vdio_irqon(int vd, enum dpio32port port) Schaltet den Pattern Interrupt scharf.

5.5 vdio_irqoff() - Disable Pattern Interrupt
int vdio_irqoff(int vd, enum dpio32port port); Schaltet den Pattern Interrupt aus.

5.6 vdio_irqwait() - Warte auf Pattern Interrupt
int vdio_irqwait(int vd, enum dpio32port port, int *count);

vd ist der von vdio_open() erhaltene "Handle"

port spezifiziert den 8-Bit-Port, dessen Interrupt-Semaphor abgefragt werden soll.

count zeigt auf eine Integer-Variable der Anwendung, die angibt, um wieviel der Interrupt-Zähler (Zählsemaphor) erniedrigt werden soll. Bei der Rückkehr enthält *count den verbleibenden Stand des Interrupt-Zählers. Der Zählsemaphor wird jedesmal um 1 erhöht, wenn die betreffende Bitgruppe einen Pattern-Match-Interrupt verursacht. Beim Aufruf von vdio_irqwait() wird vom Zählsemaphor der durch *count angegebene Betrag subtrahiert. Falls *count größer als der augenblickliche Stand des Zählers ist wird der Zählsemaphor nur auf 0 gesetzt. Dann wartet vdio_irqwait() noch, bis der nächste Interrupt für diesen Port eintrifft und kehrt dann zum Aufrufer zurück.

Anmerkung: vdio_irqwait() läßt sich auch zum einfachen Abfragen des Zählsemaphors nutzen, indem man einen Wert von count = 0 übergibt.

RETURN Im Fehlerfall wird -1 geliefert mir entspr. errno-Code, sonst 0.

5.7 vdio_write() - Schreiben 32 Bit breit, mit Maske
int vdio_write(int vd, unsigned long data, unsigned long mask); Schreibt in alle Output-Ports.

vd ist der von vdio_open() erhaltene "Handle".

Die Bytes von data werden in die Ports des VME-DPIO32 geschrieben. Dabei erfolgt die Zuordnung nach aufsteigender Wertigkeit in der üblichen Reihenfolge: (MSB = Kanal 32 ... LSB = Kanal 1)

Es werden nur diejenigen Bits verändert, deren korrespondierende Bits in mask gesetzt sind.

5.8 vdio_read() - Lesen aller 32 Port-Bits
int vdio_read(int vd, unsigned long *data); Liest alle Input-Ports des VME-DPIO32.

vd ist der von vdio_open() erhaltene "Handle"

***data** ist ein vom Aufrufer bereitzustellender Speicherplatz, in den der Inhalt der Input-Ports des VME-DPIO32 kopiert wird. Dabei erfolgt die Zuordnung zu den Ports nach aufsteigender Wertigkeit in der üblichen Reihenfolge: (MSB = Kanal 32 ... LSB = Kanal 1)

5.9 vdio_writebit() - Schreiben eines einzelnen Port-Bit
int vdio_writebit(int vd, int bitno, int value); Schreibt ein einzelnes Bit in einen der Output-Ports

vd ist der von vdio_open() erhaltene "Handle"

bitno ist die Kanal-Nummer des betroffenen Bits (1 bis 32)

value liefert in seinem LSB ("0" oder "1") den Wert, der auf dem ausgewählten Kanal ausgegeben wird.

5.10 vdio_readbit() - Lesen eines einzelnen Port-Bit
int vdio_readbit(int vd, int bitno, int *value); Liest ein einzelnes Bit aus einem der Output-Ports

vd ist der von vdio_open() erhaltene "Handle"

bitno ist die Kanal-Nummer des betroffenen Bits (1 bis 32)

***value** ist ein vom Aufrufer bereitzustellender Speicherplatz, in den das durch bitno ausgewählte Bit kopiert wird: Falls das Bit Null war, wird *value auf 0 gesetzt, andernfalls auf 1.

5.11 vdio_readstatus() - Lesen des Fehlerstatus der Ausgangstreiber
int vdio_readstatus(int vd, unsigned long *data);

vd ist der von vdio_open() erhaltene "Handle"

***data** ist ein vom Aufrufer bereitzustellender Speicherplatz, in den der Fehlerstatus der Ausgangstreiber des VME-DPIO32 kopiert wird. Dabei erfolgt die Zuordnung der Bytes zu den Ports nach aufsteigender Wertigkeit in der hier üblichen Reihenfolge: (MSB, Kanal 32, Port3 ... LSB, Kanal 1, Port0)

Fehler werden in folgenden Fällen gemeldet: Kurzschluß, fehlende Last, Überspannung oder Übertemperatur. Da die Ports in Gruppen zu je 4 Ports zusammengefaßt sind, sind jeweils 4 Bit pro Portgruppe gleich.

Haben die 4 Bit den Wert 0000 liegt kein Fehler vor, beim Wert 1111 ist mindestens auf einem Port ein Fehler aufgetreten, ohne daß der tatsächliche Port bekannt ist.

E.6 Die Analoge In/Out Schnittstelle VAIO16

Die Analoge Ein/Ausgabe-Schnittstelle VAIO16 bietet 16 von einander unabhängige Eingänge (A/D) und 4 Ausgänge (D/A) für Spannungen zwischen 0 und 5 Volt.
Zur Anbindung von Anwenderprogrammen steht eine Funktionen-Bibliothek "libvaio.a" zur Verfügung, die mit "**libvaio.a**" und mit "**liblynx.a**" gelinkt werden muß.
Benötigte Headerdatei: #include<vaio.h>

Bibliotheksfunktionen:

6.1 vaio_open() - Öffnen des Geräts
6.2 vaio_close() - Deaktivierung, Freigabe der Ressourcen
6.3 vaio_reset() - Restart VME-AIO16, optional neue CPU-Frequenz
6.4 vaio_getcalib() - Hole Kalibrierdaten
6.5 vaio_modefree() - Initialisierung des unsynchronisierten Betriebs
6.6 vaio_modesync() - Initialisierung der synchronen Betriebsart
6.7 vaio_start() - Start Konversion
6.8 vaio_stop() - Stop Konversion
6.9 vaio_readfree() - Lies A/D-Kanal unsynchronisiert
6.10 vaio_writefree() - Schreibe D/A-Kanal unsynchronisiert
6.11 vaio_readsync() - Lies A/D-Buffer synchron
6.12 vaio_writesync() - schreibe D/A-Buffer synchron

Bibliotheksfunktionen:

6.1 vaio_open() Öffnen des Geräts
int vaio_open("/dev/vaio")
Öffnet das durch den Gerätenamen (/dev/vaio) spezifizierten Gerät, prüft den Installationszustand, erfragt die im Kernel hinterlegten Geräteinformationsdaten (Device Info) und mappt die Register des Gerätes in den Adressraum des Prozesses. Das Mapping wird durch ein sogen. Physical Shared Memory (PSMEM) realisiert. Dazu muß die Anwendung nicht nur mit "libvaio.a", sondern auch mit "liblynx.a" gelinkt werden.

RETURN vaio_open liefert eine Zahl zurück, die für nachfolgende Funktionsaufrufe als "Handle" dient. Man beachte, dass diese Zahl kein Filedeskriptor ist, sondern ein interner Index der Bibliothek.
Im Fehler fall liefert vaio_open() den Wert -1, weitere Fehlerinformation findet sich ggf. in errno.
EBUSY Treiber bereits geöffnet
errno errno-Codes von open() oder smem_create()
Insbesondere ist es nicht gestattet, ein schon geöffnetes Gerät im gleichen Programm ein zweites Mal zu öffnen, ohne es vorher zu schließen.

6.2 vaio_close() - Deaktivierung, Freigabe der Ressourcen
int vaio_close(int va)
Ruft die close-Funktion des Treibers auf, blockiert alle Interrupts des Geräts und deallokiert alle durch die Bibliothek beanspruchten Ressourcen, entfernt die Register der VME-VAIO16 Hardware aus der Memory-Map des Prozesses und versucht schließlich das zugehörige PSMEM zu löschen.
va - der von vaio_open() erhaltene "Handle"

RETURN Im Fehlerfall wird -1 geliefert mir entspr. errno-Code, sonst 0.
EBADF va ist kein gültiger Handle von vaio_open()
errno errno-Codes von close().

CAVEAT
1. vaio_close() wird nicht automatisch beim exit() aufgerufen.
2. Das Löschen des PSMEM mißlingt wenn zu diesem Zeitpunkt noch andere Prozesse Zugriff darauf haben. Dies gilt nicht als Fehler

6.3 vaio_reset() - Restart VME-AI016, optional neue CPU-Frequenz

int vaio_reset(int va, long *cpufreq)

vaio_reset() führt einen Neustart der VME-AIO16 inklusive Selbsttest und Gewinnung der Kalibrierungsdaten durch.

va ist wie üblich der von vaio_open() gelieferte Handle für die Device-Instanz.

cpufreq bietet, je nach Wert, verschiedene zusätzliche Optionen:

(-1) System-Neustart mit Default-Parametern, stellt die Factory-Defaults wieder her.

NULL System-Neustart

sonst cpufreq wird als Pointer interpretiert, der auf einen Speicherplatz vom type long zeigt; der dort vom Aufrufer abgelegte Wert wird als neue Taktfrequenz (in kHz von 0 ... 25 000) für den lokalen Prozessor der VME-AIO16 übernommen. Auf *cpufreq wird die tatsächliche device-interne Frequenz "tifreq" zurückgeliefert.

RETURN Im Fehlerfall wird -1 geliefert mit entspr. errno-Code, sonst 0.

6.4 vaio_getcalib() - Hole Kalibrierdaten

int vaio_ca1ib(int va, struct vaio_calib *calibdata)

6.5 vaio_modefree() - Initialisierung des unsynchronisierten Betriebs

int vaio_modefree(int va, struct vaio_freeset *setupdata)

DESCRIPTION

vaio_modefree() konfiguriert das VME-AIO16 Device, den Treiber und die library für unsynchronisierten A/D- und D/A-Betrieb. Speziell werden hier die benutzten Kanäle und die Frequenz der Wandlung festgelegt. In dieser Betriebsart erzeugt das Gerät keine Interrupts, vielmehr werden A/D- und D/A-Wandler mit (typisch) hoher Frequenz durch einen internen Timer des Geräts getrieben. Die Wandlerzyklen starten automatisch, ohne daß der Host darauf Rücksicht nimmt. Die Anwendung darf ohne weitere Synchronisation Daten vom ADC lesen oder in den DAC schreiben. Die Verzögerung zwischen äußerem Signalpegel und den rechnerseitigen Daten hängt natürlich von der Wiederholungsfrequenz der Wandlung ab.

In einem Multi-Tasking System ist allerdings nicht zu erwarten, daß der unsynchronisierte Betrieb zu einer stabilen Abtastfrequenz (d.h. zu einem gleichmäßigen Datenfluß) führt, vielmehr ist, abhängig vom Scheduling, mit mehr oder weniger Jitter zu rechnen. Der unsynchronisierte Betrieb ist zunächst als besonders zu handhabender Testmodus gedacht. Zur Sicherung einer stabilen Tastfrequenz eignet sich eher der “synchrone” Modus (s. vaio_modesync).

va ist der von vaio_open() gelieferte "Handle".

setupdata zeigt auf eine vom Aufrufer bereitzustellende Datenstruktur; mit der die folgenden Parameter-einstellungen spezifiziert werden können (zulässige Wertebereiche in Klammern):

```
vend     = (1... 16)          ; die (End)Adresse des ausgewählten Eingangskanals (A/D)
vstart   = (1... vend)                ; die (Start)Adresse des ausgewählten Eingangskanals (A/D)
vadsrv   = (AD_RAW=1,  AD_CORR=2); Arbeitsweise des A/D-Wandlers
daend    = (1. .4)            ; die (End)Adresse des ausgewählten Ausgangskanals (D/A)
dastart  = (1...daend)        ; die (Start)Adresse des ausgewählten Ausgangskanals (D/A)
cnvtime  = (20000 .... 5682511 ns)  ; Wandlungszeit der A/D- und D/A-Wandler
```

Der Wertebereich für cnvtime ist hier für die Default-Taktfrequenz der lokalen CPU der VME-AIO16 (23068 kHz) angegeben. Nicht jeder Wert dieses Bereich ist jedoch exakt erreichbar. VME-AIO16 berechnet daher die beste Näherung für cnvtime. Vaio_modefree() gibt den tatsächlich gesetzten Wert in cnvtime an den Aufrufer zurück.

Einige weitere Parameter des VME-AIO16 werden fest eingestellt

vvtrg = (IRQ_OFF=0)

vdasrv = (DA_NORMAL=0)

ldcmod = (LDAC_WITH_START_ADC=1)

RETURN Im Fehlerfall wird -1 geliefert mir entspr. Errno-Code, sonst 0.

EBADF va ist kein gültiger Handle von vaio_open().

EIO Fehler bei Kommunikation mit VME-VAIO16 Hardware (i.a. wegen falscher Parameter).
CAVEAT Diese Funktion ruft implizit vaio_stop() auf.
Für den Beginn des Wandlerbetriebs ist ein zusätzlicher Aufruf von vaio_start() nötig.

6.6 vaio_modesync() - Initialisierung der synchronen Betriebsart
int vaio_modesync(int va, struct vaio_syncset *setupdata)
DESCRIPTION
Im synchronen Betrieb wird der Buffer-Mode der VME-AIO16 genutzt, um einen kontinuierlichen Datenfluß der Wandlerdaten zwischen Host und VME-AIO16 zu erzielen. Die Interrupts der VME-AIO16 fordern jeweils früh genug den nächsten Datentransfer durch den Host an, So kann der Host seine Aufgabe korrekt erfüllen, ohne dadurch allzu streng zeitlich engagiert zu sein.

va ist wie üblich der von vaio_open() gelieferte Handle für die Device-Instanz.
setupdata zeigt auf eine vom Aufrufer bereitzustellende Datenstruktur, mit der die folgenden Parametereinstellungen spezifiziert werden können (zulässige Wertebereiche in Klammern):

```
vend    = (1... 16)          ; die (End)Adresse des ausgewählten Eingangskanals (A/D)
vstart  = (1... vend)        ; die (Start)Adresse des ausgewählten Eingangskanals (A/D)
vadsrv  = (AD_RAW=1, AD_CORR=2, AD_1SHOT=0x10, AD_CONT=0x11);  Arbeitsweise (A/D)
ad_frames     = (0.... 0x7fff)    // frames per buffer
ad_buffers    = (0.... 0x7fff)
daend         = (1. .4)           ; die (End)Adresse des ausgewählten Ausgangskanals (D/A)
dastart       = (1...daend)       ; die (Start)Adresse des ausgewählten Ausgangskanals (D/A)
vdasrv        = (DA_NORMAL=0, DA_1SHOT=1, DA_WRAP=2); Arbeitsweise (D/A)
da_frames     = (0.... Ox7fff)    // frames per buffer
da_buffers    = (0.... Ox7fff)
cnvtime       = (20000 .... 5682511 ns)  ; Wandlungszeit der A/D- und D/A-Wandler
```

6.7 vaio_start() - Start Konversion
int vaio_start(int va, enum trigmod trigger)
DESCRIPTION
vaio_Start() aktiviert die A/D- und D/A-Konversion.

va ist der von vaio_open() gelieferte Handle für die Device-Instanz.
Das enumerative Argument **trigger** kann 3 verschiedene Werte annehmen:
ONCE der device-interne Parameter trigmod wird auf SOFTWARE gesetzt und es wird genau ein Konversionszyklus gestartet.
EXTERN trigmod wird auf EXTERN gesetzt: wenn ein Trigger-Impuls am externen Triggereingang der VME-AIO16 erscheint, startet jeweils ein neuer Konversionszyklus.
TIMER trigmod wird auf TIMER gesetzt und der Timer auf der VME-AIO16 wird gestartet. Er aktiviert automatisch die Konversion, und zwar mit der zuvor durch vaio_modefree() bzw. vaio_modesync() gewählten Periode cnvtime.
CAVEAT Im synchronen Betrieb (nach vaio_modesync()) ist ONCE als Startmodus i.A. nicht sinnvoll.

RETURN Im Fehlerfall wird -1 geliefert mir entspr. errno-Code, sonst 0.
EIO Fehler bei Kommunikation mit VME-VAIO16 Hardware (i.a. wegen falscher Parameter).

6.8 vaio_stop() - Stop Konversion
int vaio_stop(int va)
DESCRIPTION
schaltet die durch vaio_start() gesetzten automatischen Trigger-Modi EXTERN bzw. TIMER ab. Der interne Parameter trigmod der VME-AIO16 wird auf SOFTWARE (ONCE) zurückgesetzt.
vaio_stop() löst keine weitere Konversion aus.
RETURN Im Fehlerfall wird -1 geliefert mir entspr. Errno-Code, sonst 0.
EIO Fehler bei Kommunikation mit VME-VAIO16 Hardware (i.a. wegen falscher Parameter).

6.9 vaio_readfree() - Lies A/D-Kanal unsynchronisiert
int vaio_readfree(int va, int channel, int *buffer)

DESCRIPTION
vaio_readfree() liefert die bei der letzten A/D-Konversion erhaltenen Daten des ausgewählten Kanals unter Beachtung der durch vaio_modefree() bzw. vaio_modesync() konfigurierten Parameter.

va ist der von vaio_open() gelieferte Handle für die Device-Instanz.
channel bezeichnet den A/D-Kanal, der gelesen werden soll. Es muss gelten:

vstart <= channel <= vend.

buffer zeigt auf einen vom Aufrufer bereitzustellenden Speicherplatz des Typs integer. Der gelesene Konversionswert wird auf *buffer abgelegt. In dieser unsynchronisierten Betriebsart wird nicht auf das Ende einer Konversion gewartet, sondern es wird der Wert kopiert, der gerade vorgefunden werden. Er entstammt der letzten abgeschlossenen Konversion. Abhängig vom eingestellten Verarbeitungsmodus vadsrv liefert vaio_readfree() den Rohwert adwert [channel] oder den kalibrierten Wert advac [channel].

RETURN Im Fehlerfall wird -1 geliefert mir entspr. errno-Code, sonst 0.
EBAD va ist kein gültiger Handle von vaio_open().
EINVAL channel außerhalb des zulässigen Bereichs
EIO Fehler bei Kommunikation mit VME-VAIO16 Hardware (i.a. wegen falscher Parameter).

6.10 vaio_writefree() - Schreibe D/A-Kanal unsynchronisiert
int vaio_writefree(int va, int channel, int avalue)

DESCRIPTION
vaio_write() übergibt einen Digitalwert an den D/A-Konverter des ausgewählten Kanals unter Beachtung der durch vaio_modefree() bzw. vaio_ modesync() konfigurierten Parameter.
va ist wie üblich der von vaio_open() gelieferte Handle für die Device-Instanz.
channel bezeichnet den D/A-Kanal, der gelesen werden soll. Es muss gelten:

dastart <= channel <= daend

avalue ist der neue Wert, der über den D/A-Kanal ausgegeben werden soll. In dieser unsynchronisierten Betriebsart wird nicht gewartet, sondern avalue wird in den durch channel bezeichneten Wandler gerieben, ohne ein SWDALC auszulösen. Mit der nächsten Konversion wird dann der entsprechende Analogwert ausgegeben.

RETURN Im Fehlerfall wird -1 geliefert mir entspr. errno-Code, sonst 0.
EBADF va ist kein gültiger Handle von vaio_open().
EINVAL channel außerhalb des gültigen Bereichs.
EIO Fehler bei Kommunikation mit VME-VAIO16 Hardware (i.a. wegen falscher Parameter>.

6.11 vaio_readsync() - Lies A/D-Buffer synchron
int vaio_readsync(int va, int *buffer, enum blockmode mode)
DESCRIPTION
vaio_readsync() liefert einen Datenbuffer des A/D-Konverters wie durch vaio_modesync() konfiguriert: der „älteste" verfügbare A/D-Buffer wird komplett in den Speicher der Anwendung kopiert.
va ist wie üblich der von vaio_open() gelieferte Handle für die Device-Instanz.

buffer zeigt auf einen vom Aufrufer bereitzustellenden Bereich, der genug Platz für adbuf_frames*(vend-vstart+1) Integers bieten muss. vaio_readsync() prüft, ob der Interrupt-Semaphor von VAIO_ADCIRQ mindestens einen gefüllten (konvertierten) A/D-Buffer signalisiert.
Das enumerierte Argument flg **mode** bestimmt, wie sich die Funktion verhalten soll, wenn kein A/D-Buffer auf dem Gerät gefüllt ist

BLOCKING Funktion wartet am Zählsemaphor VAIO_ADCIRQ auf Interrupt
NONBLOCKING Funktion wartet nicht, sondern liefert den Fehler errno = EAGAIN.

RETURN

>0 Anzahl der unverbrauchten Impulse im Interrupt-Zählsemaphor von VAIO_ADCIRQ. Dieser Wert signalisiert die Anzahl der verfügbaren, fertig konvertierten, aber noch nicht abgeholten A/D-Buffer. Die Zahl sollte kleiner als das konfigurierte adbuf_buffers sein. Falls nicht, liegt ein Overrun vor: die A/D-Buffer wurden schneller erzeugt als abgeholt, die zeitliche Zuordnung der gelesenen Konverterdaten ist dann unklar, es empfiehlt sich zumindest ein Neustart der Konversion durch vaio_start(), u.U. auch eine Neu-Konfigurierung mit geringerer Wandlerrate.

=0 OK

-1 Fehler, s. errno, funktionsspezifische Fehlercodes möglich,

EBADF va ist kein gültiger Handle von vaio_open().

EBUSY ein anderer Prozeß oder Thread wartet auf VAIO_ADCIRQ (nur bei flg=BLOCKING)

EINTR Das Warten auf den Interrupt-Semaphor wurde durch ein Prozess-Signal unterbrochen (nur bei flg=BLOCKING)

EAGAIN kein A/D-Buffer frei (nur bei flg=NONBLOCKING)

EIO Fehler bei Kommunikation mit VME-VAIO16 Hardware (i.a. wegen falscher Parameter).

6.12 vaio_writesync() - schreibe D/A-Buffer synchron

int vaio_writesync(int va, int *buffer, enum blockmode flg/mode)

DESCRIPTION

vaio_writesync() schreibt einen Datenbuffer des A/D-Konverters wie durch vaio_modesync() konfiguriert: der "nächste" freie D/A-Buffer wird mit den Anwenderdaten gefüllt.

va ist wie üblich der von vaio_open() gelieferte Handle für die Device-Instanz.

buffer zeigt auf vom Aufrufer bereitzustellende Daten von insgesamt dabuf_frames*(daend-dastart+1) Integers. vaio_writesync() überträgt die Daten nur, wenn mindestens ein D/A-Buffer auf der VME-AIO16 frei ist.

Das enumerierte Argument **flg** bestimmt, wie sich die Funktion verhalten soll, wenn kein D/A-Buffer auf dem Gerät frei ist:

BLOCKING Funktion wartet am Zählsemaphor VAIO_ADCIRQ auf Interrupt

NONBLOCKING Funktion wartet nicht, sondern liefert den Fehler errno=EAGAIN.

RETURN

>0 Anzahl der unverbrauchten Impulse im Interrupt-Zählsemaphor von VAIO_ADCIRQ. Dieser Wert signalisiert die Anzahl der freien D/A-Buffer. Die Zahl sollte kleiner als das konfigurierte dabuf_buffers sein. Falls nicht, liegt ein Overrun vor: die D/A-Buffer wurden schneller konvertiert als nachgefüllt. Dies kann relativ leicht zu Beginn geschehen, wenn die Konversion startet, ehe der erste D/A-Buffer gefüllt werden konnte. Es empfiehlt sich daher, mindestens einen D/A-Buffer vorab zu füllen, ehe man die automatische Konversion (TIMER oder EXTERN) startet.

=0 OK

-1 Fehler, s. errno, funktionsspezifische Fehlercodes möglich

EBADF va ist kein gültiger Handle von vaio_open().

EIO Fehler bei Kommunikation mit VME-VAIO16 Hardware (i.a. wegen falscher Parameter).

EAGAIN kein D/A-Buffer bereit (nur bei flg=NONBLOCKING)

EBUSY ein anderer Prozeß oder Thread wartet auf VAIO_ADCIRQ (nur bei flg=BLOCKING)

EINTR Das Warten auf den Interrupt-Semaphor wurde durch ein Prozess-Signal unterbrochen (nur bei flg=BLOCKING)

Anschlußstecker DPIO32 Digitale Input/Output-Signale

(Übergabemodul I/O-Stecker P2 Phönix FLKM64 oder FLKMS64)

Pin	Signal	Pin	Signal
1	Vcc1	2	GNDl
3	I/O_1	4	I/O_2
5	Vcc1	6	GNDl
7	I/O_3	8	I/O_4
9	Vcc2	10	GND2
11	I/O_5	12	I/O_6
13	Vcc2	14	GND2
15	I/O_7	16	I/O_8
17	Vcc3	18	GND3
19	I/O_9	20	I/O_10
21	Vcc3	22	GND3
23	I/O_11	24	I/O_12
25	Vcc4	26	GND4
27	I/O_13	28	I/O_14
29	Vcc4	30	GND4
31	I/O_15	32	I/O_16
33	Vcc5	34	GNDS
35	I/O_17	36	I/O_18
37	Vcc5	38	GND5
39	I/O_19	40	I/O_20
41	Vcc6	42	GND6
43	I/O_21	44	I/O_22
45	Vcc6	46	GND6
47	I/O_23	48	I/O_24
49	Vcc7	50	GND7
51	I/O_25	52	I/O_26
53	Vcc7	54	GND7
55	I/O_27	56	I/O_28
57	Vcc8	58	GND8
59	I/O_29	60	I/O_30
61	Vcc8	62	GND8
63	I/O_31	64	I/O_32

Beschreibung der Signale siehe 3.1.4

Anschlußstecker AIO16 Analoge Input/Output-Signale

(I/O-Stecker P2) Belegung des 64poligen Übergabemoduls

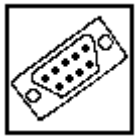
Pin	Signal	Pin	Signal
1	IN 01-	2	GND A
3	IN 01+	4	OUT 4
5	IN 02-	6	GND A
7	IN 02+	8	OUT 3
9	IN 03-	10	GND A
11	IN 03+	12	OUT 2
13	IN 04-	14	GND A
15	IN 04+	16	OUT 1
17	IN 05-	18	GND A
19	IN 05+	20	GND A
21	IN 06-	22	:
23	IN 06+	24	:
25	IN07-	26	:
27	IN 07+	28	:
29	IN 08-	30	:
31	IN 08+	32	:
33	IN 09-	34	:
35	IN 09+	36	:
37	IN 10-	38	:
39	IN 10+	40	:
41	IN 11-	42	:
43	IN 11+	44	:
45	IN 12-	46	:
47	IN 12+	48	GND A
49	1N13-	50	EX+
51	IN 13+	52	--
53	IN 14-	54	EX-
55	IN 14+	56	--
57	IN 15-	58	--
59	IN 15+	60	--
61	IN 16-	62	--
63	IN 16+	64	--

I/O - Stecker: Federleiste nach DIN41612 Bauform C64/a+c

-- ... Signal nicht angeschlossen

I_{max} per Pin: 1.0 A

Steckerbelegung



E.6 Der UART

Aufbau und Funktionsweise

Der UART 8250/16450 (Universal Asynchronous Receiver and Transmitter) hat eine Menge Funktionen ist ein recht komplexer Baustein. In Abb. 1 ein Blockdiagramm des inneren Aufbaus.
Der UART besitzt ein Empfängerteil, in Abb. 10.21 oben, und ein Senderteil, in der Abbildung. Eine Besonderheit des 8250/16450 ist, daß Sender und Empfänger mit unterschiedlichen Baudraten betrieben werden können.

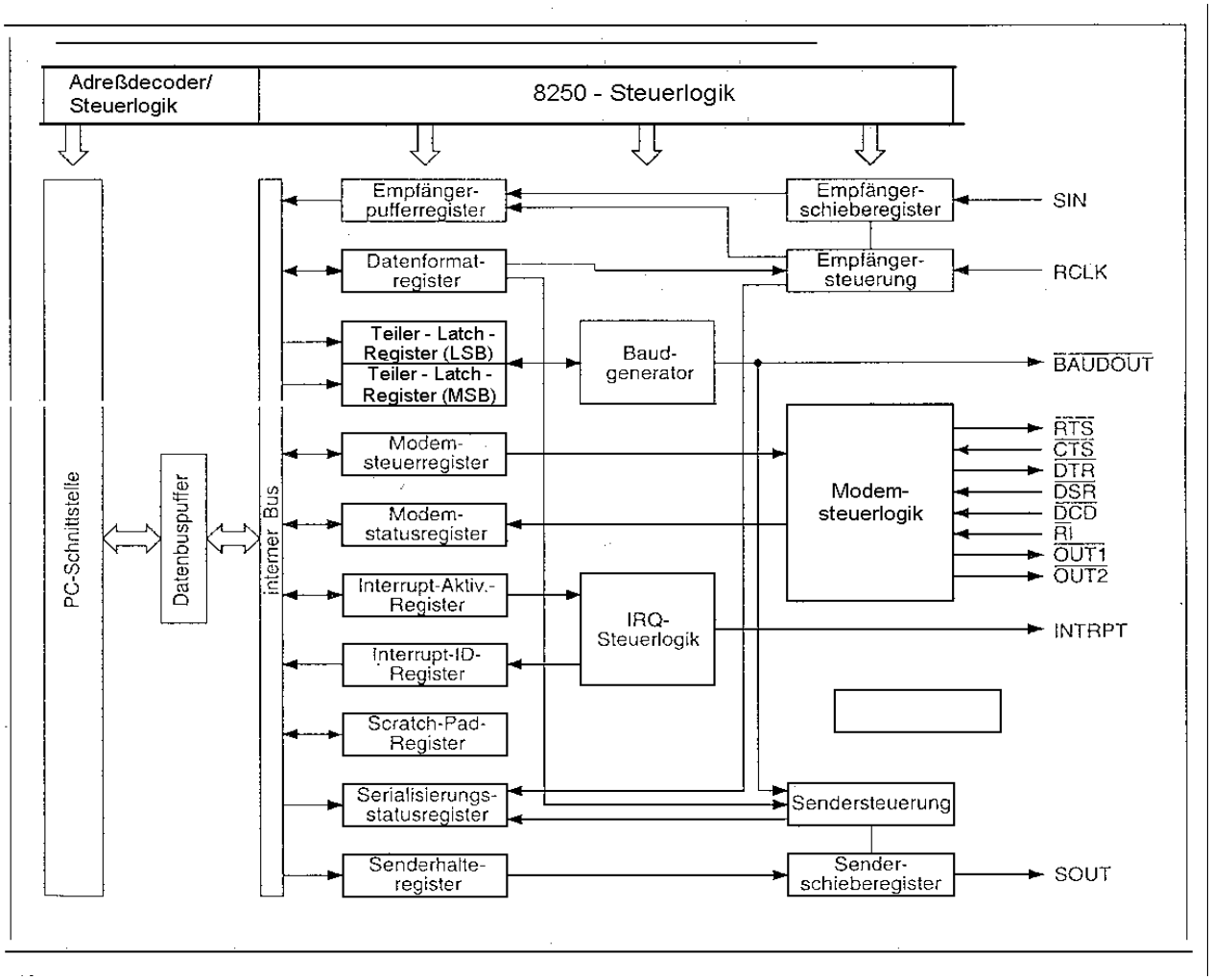


Abb. 1: Blockdiagramm des UART 8250/16450

Schnittstelle	Basisadresse	IRQ
COM1	3f8h	IRQ4
COM2	2f8h	IRQ3
COM3	3eSh	IRQ4*)
COM4	2e8h	IRQ3*)

*) oder nur Polling-Betrieb

Tab. 5: Basisadressen und IRQ-Belegungen von COMx

Register	**Offset**	DLAB	A2	A1	A0
Empfängerpufferregister	00h	0	0	0	0
Senderhalteregister					
Interrupt-Aktivierungsregister	01h	0	0	0	1
Interrupt-Identifizierungsregister	02h	-	0	1	0
Datenformatregister (Leitungssteuerregister)	03h	-	0	1	1
Modemsteuerregister (RS-232-Ausgabe)	04h	-	1	0	0
Serialisierungsstatusregister (Leitungsstatusregister)	05h	-	1	0	1
Modemstatusregister (RS-232-Eingabe>)	06h	-	1	1	0
Scratch-Pad-Register	07h	-	1	1	1
Teiler-Latsch-Register (LSB)	00h	1	0	0	0
TeiIer-Latch-Register (MSB)	0lh	1	0	0	1
DLAB=TeiIer-Latch-Zugriffsbit (Divisor Latch Access Bit)					

Tab. 10.6: Registeradressen des UART 8250

Empfängerpufferregister

7							0
E7	E6	E5	E4	E3	E2	E1	E0

E7. . .EO: Empfangsbit 0 bis 7

Senderhalteregister

7							0
S7	S6	S5	54	S3	S2	S1	S0

S7. .S0: Sendebit 0 bis 7

Abb. 3: Empfängerpuffer und Senderhalteregister (Offset 00h

7							0
0	0	0	0	SINP	ERBK	TBE	RxRD

SINP: RS-232-Eingabe
1=Interrupt bei Zustandsänderung einer RS-232-Eingangsleitung
0=kein Interrupt

ERBK: Fehler und Break (Error & Break)
1=Interrupt bei Paritäts-, Überlauf-, Framing-Fehler oder BREAK
0=kein Interrupt

TBE: Übertragungspuffer leer (Transmitter-Buffer Empty)
1=Interrupt, wenn Übertragungspuffer leer
0=kein Interrupt

RxRD: Daten empfang (Received Data Ready)
1=Interrupt, wenn ein Byte im Empfängerpufferregister bereitsteht
0=kein Interrupt

Abb. 4: Interrupt-Aktivierungsregister (Offset 01h)

7							0
0	0	0	0	0	ID1	ID0	PND

ID1, ID0 Identifizierungsbits
- 00 = Änderung eines RS-232-Eingangssignals.(Priorität 3)
- 01 = Übertragungspuffer leer (Priorität 2)
- 10 = Daten empfangen (Priorität 1)
- 10= Daten empfangen (Priorität 1)
- 11 = Serialisierungsfehler oder BREAK (Priorität 0)

0 = höchste Priorität
3 = niedrigste Priorität

PND Pending-Bit
1=kein Interrupt anhängig
0=Interrupt anhängig,

ID1, ID0 = identifizieren Ursache

Abb. 5: Interrupt-Identifizierungsregister (Offset 02h)

lnterrupt-Quelle	ID1	ID0	**Interrupt löschen durch**
Änderung eines RS~232c-Eingangssignals	0	0	RS-232C-Statusregister lesen
Senderhalteregister leer	0	1	Datenbyte in Senderhalteregister schreiben oder lnterrupt~ldentifizierungsregister lesen
Datenbyte im Empfängerpufferregister	1	0	Datenbyte aus Empfängerpufferregister lesen
Empfangsfehler oder Break	1	1	Lesen des Serialisierungsstatusregisters

Tab. 7: Anhängige Interrupts löschen

7							0
DLAB	BRK	PAR2	PAR1	PAR0	STOP	DAB1	DAB0

DLAB: Divisor-Latch Access-Bit
1=Zugriff auf Teiler-Latch
0=Zugriff auf Empfangs-/Senderegister und Interrupt-Aktivierungsregister
BRK: BREAK
1=ein 0=aus
PAR2, PAR1, PARO: Parität
000--keine
001=ungerade 011=gerade
101=mark 111=space
STOP: Anzahl der Stopbit
1=2 Stopbit 0=1 Stopbit
DAB1, DABO: Anzahl der Datenbit
00 = 5 Datenbit 01 = 6 Datenbit
10 = 7 Datenbit 11 = 8 Dastenbit

Abb. 6: Datenformatregister (Offset 03h)

7							0
0	0	0	LOOP	OUT2	OUT1	RTS	DTR

LOOP Rückkoppelungsprüfung
(Loopback-Prüfung)
1=aktiv 0=Normal betrieb
OUT2: Vielzweckausgang 2
(General-Purpose-Output 2): im PC Master-Interrupt-Aktivierungsbit
1=aktiv 0=deaktiviert
OUT1: Vielzweckausgang 1
(General-Purpose-Output 1):
1=aktiv 0=deaktiviert

RTS: Request-TO-Send
1=aktiv 0=deaktiviert
DTR: Data-Terminal-Ready
1=aktiv 0=deaktiviert

Abb. 7: Modemsteuerregister (Offset 04h)

7							0
0	TXE	TBE	BREK	FRM	PARF	ÜBLF	RxRD

TXE: Sender leer (Transmitter Empty)
1=kein Byte im Senderhalteregister und Senderschieberegister
0=ein Byte im Senderhalteregister oder Senderschieberegister
TBE: Senderhalteregister leer
(Transmitter-Buffer-Empty)
1=kein Byte im Senderhalteregister
0=ein Byte im Senderhalteregister
BREK: Break 1=erfaßt 0=kein Break
FRMF: Framing-Fehler
1=Fehler aufgetreten
0=kein Framing-Fehler
PARF: Paritätsfehler
1=Fehler aufgetreten
0=kein Paritätsfehler
ÜBLF: Überlauffehler
1=Fehler aufgetreten
0=kein Überlauffehler
RXRD:Empfangsdaten bereit
1=Empfangsdaten im Empfängerpufferregister
0=keine Empfangsdaten

Abb. 8: Serialisierungsstatusregister (Offset 05h)

7							0
DCD	RI	DSR	CTS	DDCD	DRI	DDSR	DCTS

DCD: Data-Carrier-Detect
1=DCD aktiv 0=DCD inaktiv

RI: Ring-Indicator
1=RI aktiv 0=RI inaktiv

DSR: Data-Set-Ready
1=DSR aktiv 0=DSR inaktiv
CTS: Clear To Send
1=CTS aktiv 0=CTS inaktiv

DDCD: Delta-Data-Carrier-Detect
1=DCD seit letztem Lesevorgang verändert
0=DCD gleich geblieben
DRI: Delta-Ring-Indicator
1=RI seit letztem Lesevorgang verändert
0=RI gleich geblieben
DDSR: Delta-Data-Set-Ready
1=DSR seit letztem Lesevorgang verändert
0=DSR gleich geblieben
DCTS: Delta-Clear-To-Send
1=CTS seit letztem Lesevorgang verändert
0=CTS gleich geblieben

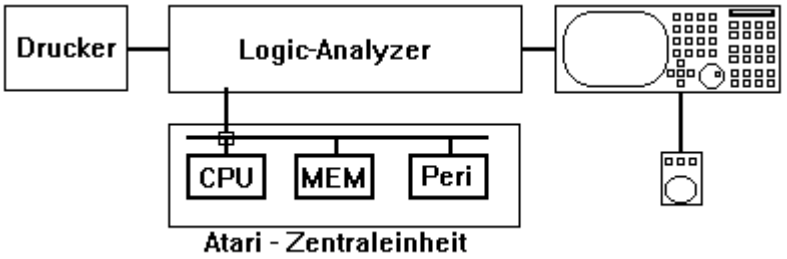
Abb. 9: Modemstatusregister (Offset 06h)

7							0
SPR7	SPR6	SPR5	SPR4	SPR3	SPR2	SPR1	SPR0

Abb. 10: Scratch-Pad-Register (Offset 07h)

E.7 Der Logikanalysator PM 3585

Die Benutzerschnittstelle besteht aus
- Eingabemöglichkeiten über
- eine Tastatur mit Funktions- und alphanumerischen Tasten,
- einen Drehknopf und
- eine Maus,
- ein Diskettenlaufwerk (im Labor ist es auf ReadOnly eingestellt)
- einer Bildschirmanzeige und
- einem Drucker für Hardcopies der Bildschirmanzeige.



Die Tastatur umfaßt mehrere Felder:
1. Funktionsauswahltasten: Configure, Format, Trace, I/O, Display, Print, Start und Stop
2. Die alphabetische und die numerische Tastatur, die praktisch nur für Dateinamen beim I/O verwendet wird.
3. Cursortasten

Alle Funktionstasten können durch Anklicken von Feldern im Display mit der Maus ersetzt werden.

Mit dem Drehknopf können Parameter im Parameterfeld des Display (s.w.u.) schrittweise variiert werden. Die Auswahl entsprechender Parameter erfolgt durch Cursortasten oder mit der Maus

Die Bildschirmanzeige umfaßt verschiedene Funktionen: Configuration, Format, Trace, I/O, Display. Hiervon werden im Labor nur die Funktionen I/O und Display verwendet.

Mit der Funktion I/O können Messungen von einer Diskette abgerufen oder auf ihr gespeichert werden. Diese Möglichkeit wird im Labor nur ausnahmsweise genutzt.

Im Display können 5 verschiedene Darstellungsarten gewählt werden (Timing/State -Button)
1. Timing - Kurve
2. Timing - Liste
3. State - Kurve
4. State - Liste mit "Disa": AUS
5. State - Liste mit "Disa": EIN (Disa = Disassembling)

Das Display ist in 3 Bereich unterteilt:
1. Statuszeile: CAPS, ON/OFF, Funktion, Datum, Zeit
2. Parameterfelder je nach Darstellungsart:
   Teilung (Tlg), X, R, S-Cursor
   Timing/State, Y, Dreh, Mod, R-S,
   Scroll-Balken für den Ausschnitt aus der Gesamtmessung (mit der Maus veränderbar)
3. Anzeigefeld für Kurven oder Listen
   (Die Anzeige kann über das Menü FORMAT eingestellt werden, für die vorgegebene Aufgabe sollte hier nichts verändert werden)

Zur **Durchführung einer Messung** ist
- ein Zeitmaßstab zu wählen (dieser ist voreingestellt und sollte auch verändert werden)
- ein Triggermuster zum konfigurieren (dieses ist voreingestellt und soll nicht verändert werden)
- der Start- und dann der Stopp-Taster zu betätigen.

Zur **Ausgabe eines Display** auf dem Drucker ist einfach die Taste "PRINT" zu betätigen
(während des Ausdruckens sollen keine Eingaben getätigt werden !!)

E.8 Motorola 68000 Instruction Set.

Instruction	Description	Assembler Syntax	Data Size	Condition Codes X N Z V C
ABCD	Add BCD with extend -(Ax),-(Ay)	Dx,Dy	B--	* U * U *
ADD	ADD binary <ea>,Dn	Dn,<ea>	BWL	* * * * *
ADDA	ADD binary to An	<ea>,An	-WL	- - - - -
ADDI	ADD Immediate	#x,<ea>	BWL	* * * * *
ADDQ	ADD 3-bit immediate	#<1-8>,<ea>	BWL	* * * * *
ADDX	ADD eXtended -(Ay),-(Ax)	Dy,Dx	BWL	* * * * *
AND	Bit-wise AND Dn,<ea>	<ea>,Dn	BWL	- * * 0 0
ANDI	Bit-wise AND with Immediate	#<data>,<ea>	BWL	- * * 0 0
ASL	Arithmetic Shift Left	#<1-8>,Dy Dx,Dy <ea>	BWL	* * * * *
ASR	Arithmetic Shift Right	...	BWL	* * * * *
Bcc	Conditional Branch	Bcc.S <label> Bcc.W <label>	BW-	- - - - -
BCHG	Test a Bit and CHanGe	Dn,<ea> #<data>,<ea>	B-L	- - * - -
BCLR	Test a Bit and CLeaR	...	B-L	- - * - -
BSET	Test a Bit and SET	...	B-L	- - * - -
BSR	Branch to SubRoutine	BSR.S <label> BSR.W <label>	BW-	- - - - -
BTST	Bit TeST #<data>,<ea>	Dn,<ea>	B-L	- - * - -
CHK	CHecK Dn Against Bounds	<ea>,Dn	-W-	- * U U U
CLR	CLeaR	<ea>	BWL	- 0 1 0 0
CMP	CoMPare	<ea>,Dn	BWL	- * * * *
CMPA	CoMPare Address	<ea>,An	-WL	- * * * *
CMPI	CoMPare Immediate	#<data>,<ea>	BWL	- * * * *
CMPM	CoMPare Memory	(Ay)+,(Ax)+	BWL	- * * * *
DBcc	Looping Instruction	DBcc Dn,<label>	-W-	- - - - -
DIVS	DIVide Signed	<ea>,Dn	-W-	- * * * 0
DIVU	DIVide Unsigned	<ea>,Dn	-W-	- * * * 0
EOR	Exclusive OR	Dn,<ea>	BWL	- * * 0 0
EORI	Exclusive OR Immediate	#<data>,<ea>	BWL	- * * 0 0
EXG	Exchange any two registers	Rx,Ry	--L	- - - - -
EXT	Sign EXTend	Dn	-WL	- * * 0 0
ILLEGAL	ILLEGAL-Instruction Exception	ILLEGAL		- - - - -
JMP	JuMP to Affective Address	<ea>		- - - - -
JSR	Jump to SubRoutine	<ea>		- - - - -
LEA	Load Effective Address	<ea>,An	--L	- - - - -
LINK	Allocate Stack Frame	An,#<displacement>		- - - - -
LSL	Logical Shift Left	Dx,Dy #<1-8>,Dy <ea>	BWL	* * * 0 *
LSR	Logical Shift Right	...	BWL	* * * 0 *
MOVE	Between Effective Addresses	<ea>,<ea>	BWL	- * * 0 0
MOVE	To CCR	<ea>,CCR	-W-	I I I I I
MOVE	To SR	<ea>,SR	-W-	I I I I I
MOVE	From SR	SR,<ea>	-W-	- - - - -
MOVE	USP to/from Address Register	USP,An An,USP	--L	- - - - -
MOVEA	MOVE Address	<ea>,An	-WL	- - - - -

```
MOVEM    MOVE Multiple                  <register list>,<ea>  -WL   - - - - -
                                        <ea>,<register list>
MOVEP    MOVE Peripheral                Dn,x(An)     -WL           - - - - -
                                        x(An),Dn
MOVEQ    MOVE 8-bit immediate        #<-128.+127>,Dn --L           - * * 0 0
MULS     MULtiply Signed                <ea>,Dn      -W-           - * * 0 0
MULU     MULtiply Unsigned              <ea>,Dn      -W-           - * * 0 0
NBCD     Negate BCD                     <ea>         B--           * U * U *
NEG      NEGate                         <ea>         BWL           * * * * *
NEGX     NEGate with eXtend             <ea>         BWL           * * * * *
NOP      No OPeration                NOP                           - - - - -
NOT      Form one's complement          <ea>         BWL           - * * 0 0
OR       Bit-wise OR                    <ea>,Dn      BWL           - * * 0 0
                                        Dn,<ea>
ORI      Bit-wise OR with Immediate     #<data>,<ea> BWL           - * * 0 0
PEA      Push Effective Address         <ea>         --L           - - - - -
RESET    RESET all external devices  RESET                         - - - - -
ROL      ROtate Left                    #<1-8>,Dy    BWL           - * * 0 *
                                        Dx,Dy
                                        <ea>
ROR      ROtate Right                   ...          BWL           - * * 0 *
ROXL     ROtate Left with eXtend        ...          BWL           * * * 0 *
ROXR     ROtate Right with eXtend       ...          BWL           * * * 0 *
RTE      ReTurn from Exception          RTE                        I I I I I
RTR      ReTurn and Restore             RTR                        I I I I I
RTS      ReTurn from Subroutine         RTS                        - - - - -
SBCD     Subtract BCD with eXtend       Dx,Dy        B--           * U * U *
                                        -(Ax),-(Ay)
Scc      Set to -1 if True, 0 if False  <ea>         B--           - - - - -
STOP     Enable & wait for interrupts #<data>                       I I I I I
SUB      SUBtract binary                Dn,<ea>      BWL           * * * * *
                                        <ea>,Dn
SUBA     SUBtract binary from An        <ea>,An      -WL           - - - - -
SUBI     SUBtract Immediate             #x,<ea>      BWL           * * * * *
SUBQ     SUBtract 3-bit immediate       #<data>,<ea> BWL           * * * * *
SUBX     SUBtract eXtended           Dy,Dx           BWL           * * * * *
                                        -(Ay),-(Ax)
SWAP     SWAP words of Dn               Dn           -W-           - * * 0 0
TAS      Test & Set MSB & Set N/Z-bits  <ea>         B--           - * * 0 0
TRAP     Execute TRAP Exception         #<vector>                  - - - - -
TRAPV    TRAPV Exception if V-bit Set TRAPV                        - - - - -
TST      TeST for negative or zero      <ea>         BWL           - * * 0 0
UNLK     Deallocate Stack Frame         An                         - - - - -

--------------------------
Symbol   Meaning
------   -------
   *     Set according to result of operation
   -     Not affected
   0     Cleared
   1     Set
   U     Outcome (state after operation) undefined
   I     Set by immediate data

<ea>     Effective Address Operand
<data>   Immediate data
<label>  Assembler label
<vector> TRAP instruction Exception vector (0-15)
<rg.lst> MOVEM instruction register specification list
<displ.> LINK instruction negative displacement
... Same as previous instruction
```

```
-------------------------
Addressing Modes           Syntax
----------------------     --------------------------
Data Register Direct Dn
Address Register Direct    An
Address Register Indirect  (An)
Address Register Indirect with Post-Increment  (An)+
Address Register Indirect with Pre-Decrement   -(An)
Address Register Indirect with Displacement    w(An)
Address Register Indirect with Index     b(An,Rx)
Absolute Short       w
Absolute Long        l
Immediate #x
Status Register      SR
Condition Code Register    CCR
Program Counter with Displacement     w(PC)
Program Counter with Index b(PC,Rx)

Legend
---------
   Dn     Data Register     (n is 0-7)
   An     Address Register  (n is 0-7)
    b     08-bit constant
    w     16-bit constant
    l     32-bit constant
    x     8-, 16-, 32-bit constant
   Rx     Index Register Specification, one of:
    -  Dn.W  Low 16 bits of Data Register
    -  Dn.L   All 32 bits of Data Register
    -  An.W  Low 16 bits of Address Register
    -  An.L   All 32 bits of Address Register

--------------------------------------------------
Condition Codes for Bcc, DBcc and Scc Instructions.
--------------------------------------------------
Condition Codes set after CMP D0,D1 Instruction.

Relationship          Unsigned            Signed
------------          --------            ------

D1 <  D0  CS - Carry Bit Set              LT - Less Than
D1 <= D0  LS - Lower or Same              LE - Less than or Equal
D1  = D0  EQ - Equal (Z-bit Set)          EQ - Equal (Z-bit Set)
D1 != D0  NE - Not Equal (Z-bit Clear)    NE - Not Equal (Z-bit Clear)
D1 >  D0  HI - HIgher than                GT - Greater Than
D1 >= D0  CC - Carry Bit Clear            GE - Greater than or Equal
          PL - PLus (N-bit Clear)         MI - Minus (N-bit Set)
          VC - V-bit Clear (No Overflow)  VS - V-bit Set (Overflow)
          RA - BRanch Always
DBcc Only  F - Never Terminate (DBRA is an alternate to DBF)
           T - Always Terminate
Scc Only  SF - Never Set
          ST - Always Set
 -------------------------
Parts from "Programming the 68000" by Steve Williams. (c) 1985 Sybex Inc.
Parts from BYTE Magazine article.

Compiled by Diego Barros.   e-mail : alien@zikzak.apana.org.au
Revision 2.1                22 May, 1994
```

E.9 Die Graphik-Bibliothek CSLIB

Die Graphik-Bibliothek CSLIB wurde von Eric Roberts entwickelt und ist im Internet beim Verlag Addison & Wesley verfügbar (HTTP://www.aw.com/compter.science/)
Auf der Benutzeroberfläche X11 können hiermit einfache Graphikausgaben durchgeführt werden.
Folgende Basisfunktionen werden angeboten:

InitGraphics(double xw, double yw);	diese Grundfunktionen muß zu Beginn aufgerufen werden, um ein Graphikfenster der Größe "xw * yw" (in Zoll) zu erstellen.
MovePen(double x, double y);	bewegt den Graphikzeiger zum Punkt mit den Koordinaten (x, y) in Zoll ohne zu zeichnen.
DrawLine(double dx, double dy);	zeichnet eine Linie von aktuellen Standpunkt zu einem Punkt mit den (relativen !) Abständen "dx" und "dy".
DrawTextString(string s);	schreibt einen Text "s" ab der aktuellen Position und bewegt den Graphikzeiger entsprechend.
Pause(int s);	wartet s Sekunden mit der Aktualisierung der Anzeige:

Weitere Funktionen sind implementiert, funktionieren aber nicht immer zuverlässig !!!!!!

Für ihre Einbindung in ein "C"-Programm sind folgende Bibliotheken erforderlich:

/xxx/cslib.a	diese Bibliothek von Grundfunktionen muß für jede Rechnerplattform neu erstellt werden.
/usr/lib/libX11.a	diese Bibliothek von Grundfunktionen muß für jede Rechnerplattform vorhanden sein, wenn die Benutzeroberfläche X11 installiert ist.
/usr/lib/libm.a	diese Bibliothek von Grundfunktionen muß für jede Rechnerplattform vorhanden sein.

Zur Vereinfachung wurde ein Shellscript "gccx" installiert, das eine Variante des Kommandos "gcc" darstellt und alle erforderlichen Bibliotheken lädt.
Im Kern enthält es folgende Anweisung: gcc -g -I$CSLIB $$* $(CSLIB -lX11 -lm)
Es wird wie ein gcc-Kommando aufgerufen, z.B. gccx -o haus haus.c
Im Quellprogramm müssen keine besonderen Header-Dateien für die Graphik angegeben werden

Auf den LynxOS-Rechnern im Realzeitlabor sind die notwendigen Bibliotheken auf folgenden Pfaden zu erreichen:
/usr/lib/libX11.a
/usr/lib/libm.a
/usr/local/lib/cslib.a

E.10 Die Digitale Fourier Transformation (DFT) FFTW

Die schnelle digitale Fourier-Transformation FFTW (Fast Fourier Transform West) ermöglicht eine Reihe von Fourier-Transformationen, von denen nur eine (III) hier verwendet werden soll:

I one-dimensional complex transforms (FFTW),
II multi-dimensional complex transforms (FFTWND),
III one-dimensional transforms of real data (RFFTW),
IV multi-dimensional transforms of real data (RFFTWND).

Die eindimensionale Fourier-Transformation von reellen Zahlen (RFFTW)

Ein typisches Programmsegment sieht etwa so aus:

```
#include <rfftw.h>
........
fftw_real in[N], out[N], power_spectrum[N/2+1]; /* N=2^k
                                                  besonders günstig!*/
rfftw_plan p;
int k;
...
p = rfftw_create_plan(N, FFTW_REAL_TO_COMPLEX, FFTW_ESTIMATE);
                                            /*einmalige Vorbereitung*/
...
rfftw_one(p, in, out);                /* die aktuelle Transformation*/
.....
power_spectrum[0] = out[0]*out[0];                  /* DC component */
for (k = 1; k < (N+1)/2; ++k)               /* (k < N/2 rounded up) */
power_spectrum[k] = out[k]*out[k] + out[N-k]*out[N-k];
if   (N % 2 == 0)                                     /* N is even */
power_spectrum[N/2] = out[N/2]*out[N/2];           /* Nyquist freq. */
...
rfftw_destroy_plan(p);                /* Freigabe von Speicherplatz */
...
```

Was rfftw_one(p, X, Y) wirklich berechnet ist: $Y_i = \sum_{j=0}^{n-1} X_j \, e^{-2pij\sqrt{-1}/n}$

Der Ausgabe-Vektor Y enthält dann ein hermitisch komplex konjugiertes Feld der unnormalisierten (!!) Spektralkomponenten mit den Realteilen r_i und den Imaginärteilen i_i in folgender Anordnung:

$r_0, r_1, \ldots r_{N/2}, i_{(N+1)/2\,-1}, \ldots i_2, i_1$

Diese Komponenten sind noch um den Faktor $\sqrt{n}$ zu normalisieren !!!!

Achtung:
Der Eingabe-Vektor X wird hierbei zerstört !!!!!!!!!!!!!!!!!!!!!!!!

Zur Einbindung in Anwenderprogrammen sind folgende Bibliotheken notwendig:
FFTW, RFFTW und MATH Bibliothek;
auf UNIX-Systemen mit: **-lrfftw -lfftw –lm**

PROGRAMMKENNDATEN In Anlehnung an DIN 66 230	Entwickler:
Programmname(n)	
	Programmsystem
Variantenbezeichnung:	Versionsbezeichnung:
erste Version freigegeben am:	aktuelle Version freigegeben am:
Deskriptoren (Schlagworte)	
Aufgabe (Kurzbeschreibung, ggf. Beispiele, fachbezogene Ein- und Ausgaben Methoden Vorschriften. Besonderheiten)	

DIN 66230

Gerätebedarf (Hardware)					
Speicherbedarf (im Arbeitsspeicher)					
Programmbedarf	Betriebssystem: Version:				
	Unterprogramme: (z.B. Treiber)				
	Sonstige Programme				
Programmiersprache	Sprache: Sprachumfang: Anzahl der Anweisungen:				
	Übersetzer: Version				
Betriebsarten	Stapelbetrieb	Dialalogbetrieb	Realzeitbetrieb	Sonstige Betriebsart	
Dateien	Bezeichnung/ Name	Datenträger / Pfad	Dateigröße / KB	Dateityp	Zugriffsart/ -berechtigung
Konventionen					
Zuständigkeiten <Anschriften>	Entwicklung				
	Vertrieb				
	Pflege				
	Weiterentwicklung				
Unterlagen (Handbücher):	Anwenderhandbuch (Seiten)				
	Entwicklerhandbuch (Seiten)				
Installierungen	Hersteller	Anlagentyp	Betriebsystem	Anzahl	
Preise/Kosten	Art des Angebots (z.B. freibleibend):				

	Kaufpreis	Mietpreis
	Installierungskosten	Wartungs- und Pflegekosten
	Schulungskosten	

Fachhochschule Frankfurt am Main

University of Applied Sciences

Fachbereich 2: Informatik und Ingenieurwissenschaften ,

Computer Science and Engineering

Labor für Realzeitsysteme (embedded systems)

Dozent / Dozentin	
Gruppe	

Gruppen-ID:	**Teilnehmerin / Teilnehmer**	Matrikelnummern
1		
2		

Versuch Nr. :	Bezeichnung:

Durchgeführt am:		Abgegeben am:	
Testat			